P. 1877/85

mikroKLAN 23-24

11-12 1985

informatyka

Porównanie języków programowania
Wprowadzenie do programometrii
Podsieć komunikacyjna MSK
Systemy zarządzania w górnictwie
Język symulacyjny PASCALSIM
Pożegnanie z mikroKLANEM

Cena 200 zł

informatyka

11-12

N r 11-12

Miesięcznik Rok XX

Listopad-grudzień 1985

Organ Komitetu Informatyki MNSZWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), mgr inż. Andrzej J. PIOTROWSKI, dr inż. Janusz ZALEWSKI

STALE WSPÓŁPRACUJĄ:

Mgr inż. Witold ABRAMOWICZ (Szwajcaria), mgr inż. Ryszard K. KOTT (Wielka Brytania), mgr inż. Teresa WILCZEK

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 1099-1300/85. Obj. 4,0 ark. druk. Nakład 6400 egz. P-67.

ISSN 0342-9951, INDEKS 36124

Cena egzemplarza 200 zł
Prenumerata roczna 1200 zł

NACZELNA ORGANIZACJA TECHNICZNA

WYDAWNICTWO

SIGMA

CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

XX lat INFORMATYKI

Porównanie języków programowania mikrokomputerów
Zbigniew Szkaradnik 2

Programowanie asemblerowe w języku fig-FORTH
Jan Bielecki 6

Wprowadzenie do programometrii
Stanisław Gasik 9

RATFOR — czyli strukturalny FORTRAN (2)
Zdzisław Płoski 13

Podsieć komunikacyjna Międzyuczelnianej Sieci Komputerowej
Edward Bieleninik, Elżbieta Hudyma, Elżbieta Kosmulska-Bochenek, Józef Lewoc, Andrzej Stanisz 15

Skomputeryzowane systemy zarządzania w górnictwie węgla kamiennego 39

35 lat Centralnego Ośrodka Informatyki Górnictwa 42

DYDAKTYKA

Zdarzeniowy język symulacyjny PASCALSIM i jego implementacja
Krystyna Koleśnik, Jan Smolira 19

mikroKLAN 20

Blaski i cienie mikroprocesora 8086
Przystawka video do komputera COMMODORE C64
Program BETA BASIC
Łuki krzywych gładkich
FORTH
Post scriptum Akademii mikroKLANU, czyli jak pisać duże programy

ZE ŚWIATA 45

Najnowsze tendencje w dziedzinie komputerowych systemów sterowania, problemy oprogramowania

TERMINOLOGIA 48

DIXIT ALGORIZMI czyli o pochodzeniu i znaczeniach słowa algorytm

RECENZJE 50

Systemy informacyjne: aspekty teoretyczne i formalne

W NASTĘPNYCH NUMERACH:

- Wiesław Nosowski i Bolesław Szomański o pakiecie symulacyjnym
- Jerzy Sukiennik o rozrachunku gospodarczym zakładowego ośrodka informatyki
- Ian Pyle o pakiecie do specyfikacji programów w Adzie
- Marek Sikora o modułowym systemie mikroprocesorowym kompatybilnym z systemem MIKROSTER
- Jerzy Zakręcki o abstrakcjach w programowaniu
- Witold Abramowicz o modularnym systemie wyszukiwawczym
- Ryszard Tuziemski o RATFORZE dla MERY 400
- Jan Bielecki o błędach implementacji w kompilatorze języka C firmy Supersoft

P. 1877/85

Spis treści rocznika INFORMATYKA 1985

ARTYKUŁY PROBLEMOWE

	nr	str.
BANASIK ZBIGNIEW — Projekt normy IEEE — arytmetyka zmiennoprzecinkowa	1	12
BIELECKI JAN — ADA — standard operacji wejścia—wyjścia	5	4
BIELECKI JAN — FORTH — definiowanie kompilatorów i instrukcji strukturalnych	7	9
BIELECKI JAN — Programowanie asemblerowe w języku fig-FORTH	11—12	6
BIELENINIK EDWARD, HUDYMA ELŻBIETA, KOSMULSKA-BOCHENEK ELŻBIETA, LEWOC JÓZEF, STANISZ ANDRZEJ — Podsieć komunikacyjna Międzyuczelnianej Sieci Komputerowej	11—12	15
BILSKI EUGENIUSZ, FRYŹLEWICZ ZBIGNIEW, HUZAR ZBIGNIEW, KALIŚ ANDRZEJ — Rozległe sieci komputerowe	9	4
BROCZKÓ PETÉR — Produkcja mikrokomputerów w 1984 r. w krajach RWPG	8	9
CELLARY WOJCIECH, WĘGLARZ JAN — Koncepcja nauczania mikroinformatyki	5	11
DZIEDZIC MAREK, PERYCZ KRZYSZTOF, WILIŃSKI JERZY — Emulator E6RM-E — przykład wykorzystania komputerów mikroprogramowanych	2	4
FURMAN MARIAN, NOWICKI ZBIGNIEW M., SOLAK JERZY — Pakiet SIMIS dla MERY 400	6	4
GACEK ADAM, KOZIELSKI STANISŁAW, STRÓŻYNA PIOTR — EW — elastycznie mikroprogramowany komputer dydaktyczny	9	22
GASIK STANISŁAW — Wprowadzenie do programometrii	11—12	9
IRLIK JACEK — Kwalifikacja prawna programu komputerowego	7	5
ISZKOWSKI WACŁAW — Przegląd mechanizmów komunikacji	3	3
ISZKOWSKI WACŁAW — Superkomputer CRAY	5	1
ISZKOWSKI WACŁAW, GRZEGORZEWICZ PAWEŁ — LOGOS — system generowania systemów operacyjnych	8	1
JABŁOŃSKI KONRAD — Język programowania BCPL		
Część 1	2	1
Część 2	3	11
KAPUŚCIK WALDEMAR — Projektowanie sieciowych baz danych na przykładzie SZBD — 1100	2	6
KARCZMARCZUK JERZY — Komputery SINCLAIRA — rachunki numeryczne w kodzie maszynowym	6	1
KIERZKOWSKI ZBIGNIEW, MAŁUSZYŃSKI JACEK — Zarządzanie współbieżną aktualizacją bazy danych		
Część 1. Funkcja oprogramowania zarządzającego	3	1
Część 2. Przegląd metod zarządzania	4	7
KOLEŚNIK KRYSTYNA, SMOLIRA JAN — Zdarzeniowy język symulacyjny PASCALSIM i jego implementacja	11—12	19
KOTT RYSZARD K. — BASIC dla mikrokomputerów — ewolucja języka	1	2
KRĘPSKI ARTUR — PASCAL MT+ dialekt PASCALA dla mikrokomputerów	1	6
LAND FRANK — Dziedzina systemów informacyjnych	4	5
LITWINIUK ANDRZEJ I. — Kompilator LOGLANU 82 dla MERY 400	3	7
MIKOŁAJCZAK BOLESŁAW, STOKŁOSA JANUSZ — Złożoność algorytmów		
Część 1. Klasyfikacja problemów	9	1
Część 2. Złożoność asymptotyczna	10	7
PERYCZ KRZYSZTOF — Egzekutor E6MR dla R-32	10	1
PŁOSKI ZDZISŁAW — RATFOR czyli strukturalny FORTRAN		
Część 1	10	4
Część 2	11—12	13
STAWOWY ADAM — MACINTOSH — nowa jakość na rynku mikrokomputerów (oprac.)	4	11
STĘPIEŃ CEZARY, SZYRKOWIEC JAROSŁAW — Wielopoziomowy system oprogramowania pamięci kasetowych PK-1	8	4
SURAJ ZBIGNIEW — Problemy nauczania informatyki w szkołach średnich	7	11
SZKARADNIK ZBIGNIEW — Porównanie języków programowania mikrokomputerów	11—12	2
WALIGÓRSKI STANISŁAW — LOGO		
Część 1. Grafika żółwia	7	1
Część 2. Słowa i listy	8	21
Część 3. Porównanie dialektów APPLE, SINCLAIR i COMMODORE 64	9	11
WILCZYŃSKI WIESŁAW — PLZ — język programowania mikroprocesorów (2)	1	9
WINIEWSKI JULIAN — Język programowania C		
Część 1	4	1
Część 2	5	7
Część 3	6	7
Problem perspektywy — **Zbigniew Gluza**	1	1
Oprogramowanie mikrokomputerów — w co się bawić? — **Janusz Zalewski**	1	2
XX lat INFORMATYKI	11—12	1
Skomputeryzowane systemy zarządzania w górnictwie węgla kamiennego	11—12	39
35 lat Centralnego Ośrodka Informatyki Górnictwa	11—12	42

DYDAKTYKA

	nr	str.
Nowy dział — **Wacław Iszkowski**	4	21
Rozwój mikroelektroniki a nauczanie — **W.I.**	4	22
Konkurs PTI na najlepsze prace dyplomowe z informatyki — **Barbara Osuchowska**	4	23
Koncepcja nauczania mikroinformatyki — **Wojciech Cellary, Jan Węglarz**	5	11
Nauczyciel — uczeń — mikrokomputer — **Emanuel Czyżo**	6	19
Problemy nauczania informatyki w szkołach średnich — **Zbigniew Suraj**	7	11
Postęp w szkoleniu informatycznym — **Andrzej Elek**	8	24
Zdarzeniowy język symulacyjny PASCALSIM i jego implementacja — **Krystyna Koleśnik, Jan Smolira**	11—12	19

SAMOTESTY

	nr	str.
II/B. Reprezentacja, przepływ i przekształcanie danych — **Adam B. Empacher, Ludwik J. Rossowski**	1	23
III/A. Podstawy sortowania zbiorów — **Adam B. Empacher, Ludwik J. Rossowski**	3	22
III/B. Metody sortowania wewnętrznego — **Adam B. Empacher, Ludwik J. Rossowski**	6	21
IV/A. Metodyka programowania — **Adam B. Empacher, Ludwik J. Rossowski**	9	26
IV/B. Rzetelność danych — **Adam B. Empacher, Ludwik J. Rossowski**	10	23

mikroKLAN

Norma GKS — **Antoni Urban**	1	13
Przystawka do ZX81 — **Rafał Pietrak**	1	15
Mikrokomputer MERITUM-I — **Zbigniew Chrześcijański, Jacek Żebrowski**	1	17
FORTH (5) — **Marek Czarzyński**	1	18
Akademia mikroKLANU — **Jakub Tatarkiewicz**		
Część 2. Silnia	1	20
Część 3. Koniunkcja	2	19
Część 4. Kursor	3	16
Część 5. Wyrażenie	4	20
Część 6. Dekada	5	14
Część 7. ACKERMAN	6	17
Część 8.	7	19
Post. scriptum Akademii mikroKLANU czyli Jak pisać duże programy	11—12	34
APPLE II c — **Adam Stawowy**	2	13
Rozszerzenie możliwości graficznych ZX81 (16 K) — **Jacek Pszona**	2	14
Współpraca z napędem dysków elastycznych — **Adam Pluta**	2	15
DS-80 MONITOR — **Grzegorz Stępień, Mirosław Doleżych, Jerzy Orkiszewski**	2	17
Komputery z drugiej ręki — **AJP**	2	18
INDUSTRIAL REAL-TIME BASIC — dla mikrokomputerów — **Wojciech Warski**	3	13
Na Węgrzech można...	3	13
Tajemnice programowych piratów — ZX SPECTRUM — **AJP**	3	15
CP/M 3+ — **Witold Woźniak**	3	18
Alfanumeryczny sterownik monitora telewizyjnego — **Grzegorz Stępień, Mirosław Doleżych**		
Część 1	3	19
Część 2	4	17
ISBN — **Jakub Tatarkiewicz**	3	20
Grafika IBM-PC — **Antoni Urban**		
Część 1	4	7
Część 2	5	18
ZX MICRODRIVE — **Jerzy Karczmarczuk, Tadeusz Wilczek, Tomasz Zieliński**	4	14
AMSTRAD CPC 464 — **Marek Górecki**	4	19
COMMODORE C 64 czy ZX SPECTRUM? — **Janusz Piechna**	5	13
Jak odzyskać kontrolę nad programem realizowanym z pamięci, której nie ma? — **AJP**	5	15
Programowy generator dźwięków AY-3-8910/12 — oprac. **AJP**	5	16
Standard RS 232C — **AJP**	5	19
Jak rozszerzyć BASIC ZX SPECTRUM		
Część 1. **Marek Górecki**	6	11
Część 2. **Marek Górecki, Andrzej J. Piotrowski**	7	18
Doświadczenia z eksploatacji IMZ-80 — **Jerzy Kurowski**	6	12
Schemat ZX SPECTRUM — **AJP**	6	13
Polska Federacja Klubów Mikroinformatycznych — **Piotr Tymochowicz**	6	17
Test numeryczny — **Jakub Tatarkiewicz**	6	18
Moduł procesora MOTOROLA 6800 — **Jerzy Orkiszewski**	7	13
SPREADSHEET dla ZX81 — **Witold Majewski**	7	15
Jeszcze o możliwościach graficznych ZX81 16 KB — **Jacek Pszona**	7	17
Układ syntezy dźwięków — **Tomasz Pietrzak**	7	20
Obliczenia symboliczne — **Jakub Tatarkiewicz**	8	13
Pamięć EPROM 2716 — **AJP**	8	14
Redagowanie i przetwarzanie tekstów programem TEKST-CSK — **Antoni Urban**	8	16
Układy z multipleksowaną szyną danych w systemie 8080 — **Jerzy Orkiszewski**	8	17
Program MONITOR dla mikroprocesora MOTOROLA 6800 — **Jerzy Orkiszewski**	8	19
Mikrokomputerowa pierwsza pomoc — **AJP**	9	13
Wyścig z czasem — **Ryszard Kurzyjamski**	9	14
Sposoby kreślenia okręgów — **Marek Dorywalski**	9	16
Przykłady procedur w języku ASSEMBLER 8080 — **Adam Pluta**		
Część 1	9	18
Część 2	10	16
Zapamiętywanie obrazu w CPC 464 — **Marek Dorywalski**	9	20
AMSTRAD CPC 664 — **Maciej Doruchowski**	10	11
Wielolinia — **Antoni Urban**	10	12
Samobieżny ZX81 — **Roland Hojczak**	10	14
Norma IEEE „w pigułce" — **Jerzy Orkiszewski**	10	18
MC 6809 — W pół kroku między 6800 a 68 000 — **Jerzy Orkiszewski**	10	19
Wokół języka FORTH — **Michał Choroszucha**	10	21
Blaski i cienie mikroprocesora 8086 — **Andrzej Piotrowski**	11—12	20
Przystawka video do komputera COMMODORE C 64 — **Roland Hojczak**	11—12	25
Program BETA BASIC — **Jerzy Krystek, Adam Mielczarek**	11—12	29
Łuki krzywych gładkich — **Antoni Urban**	11—12	30
FORTH — **Michał Choroszucha**	11—12	32

Z KRAJU

Uchwała I Walnego Zjazdu PTI w sprawie rozwoju informatyki w Polsce	1	24
Wynik z komputera — **Jerzy Kurowski**	1	25
Komputery w „N-tym" resorcie — **H. Z.**	2	10
Gra finansowa na MERZE 400 — **Mieczysław Budziński, Edward Radosiński, Leopold Szczurowski**	2	11
Obsługa techniczna sprzętu w zestawieniu ankietowym — **Janusz Maciejewski**	2	12
Nowa rodzina IBM — **A.B.E.**	3	23
57 MTP. Pogoń za mikro — **Andrzej J. Piotrowski**	5	23
RATFOR na ODRZE 1305	5	24
Mikrokomputery w Pałacu Młodzieży — **Janusz Zalewski**	6	23
Rozproszone systemy przetwarzania danych — **Julian Dębowy**	6	24
Komputerowy system układania rozkładu zajęć w szkole wyższej — **Marek Skomorowski, Ryszard Woźniak**	7	22
PTI w sprawie wprowadzania informatyki do szkół — **Prezydium ZG PTI**	7	22

XX lat INFORMATYKI

Obecny numer INFORMATYKI zamyka 20 lat istnienia czasopisma. Chociaż w działalności edytorskiej eksponowanie tej skali jubileuszu publikacji nie zawsze jest uzasadnione, to jednak w zestawieniu z zaledwie 36-letnią historią rozwoju techniki cyfrowej jest to okres dostatecznie długi, aby etap ten zaznaczyć, a przy tej okazji podzielić się okolicznościowymi refleksjami.

Gdy w ostatnim kwartale 1965 r. na krajowym rynku wydawniczym pojawił się pierwszy numer naszego czasopisma pod swą pierwotną i niezbyt atrakcyjną nazwą „Maszyny Matematyczne" minęło zaledwie szesnaście lat od uruchomienia pierwszej na świecie, a siedem lat — pierwszej polskiej elektronicznej maszyny cyfrowej. Wprawdzie współczesne tempo rozwoju informatyki wielokrotnie wzrosło i obecnie 10 lat odpowiada przemianom zachodzącym przed trzydziestu laty w czasie życia dwóch generacji ludzkich, to jednak zestawienie tych dwu porównawczych miar czasu pozwala stwierdzić, że kraj nasz ze stosunkowo niewielkim opóźnieniem w stosunku do światowej premiery maszyny cyfrowej, a znacznie wcześniej niż większość krajów naszego obozu, zdołał wejść do bardzo nielicznej grupy państw, które opanowały sztukę budowy komputerów. Jeśli uwzględnimy ówczesny bardzo skromny potencjał przemysłowy kraju oraz szczupłość kadr naukowych, których nie zdołano jeszcze uzupełnić po straszliwych stratach spowodowanych okupacyjną eksterminacją, fakt ten może dziś budzić zdumienie, a nawet niedowierzanie. Jeszcze bardziej zaskakującym faktem jest rozpoczęcie w Polsce już w 1960 r., niespełna dwa lata po uruchomieniu pierwszego komputera, jego przemysłowej produkcji. Przypomnienie tych zapomnianych i bliżej nie znanych współczesnej generacji zdarzeń, skłonić powinno do zdecydowanego przeciwstawienia się nastrojom pesymizmu, jakie często rodzą się z obserwacji naszego obecnego zapóźnienia informatycznego.

Przytoczone ze stosunkowo niedalekiej przeszłości przykłady potwierdzają bowiem dobrze znaną prawdę, że o pozycji społeczeństwa decydują nie tyle jego zasoby materialne, tradycje techniczne czy liczba ludności, ile jego prężność intelektualna, która na szczęście jest jedną z naszych głównych dodatnich cech narodowych. Prężność ta jest właśnie atutem, który obecnie decyduje o efektywnym spożytkowaniu olbrzymich i tak dziś łatwo dostępnych wskutek taniości sprzętu komputerowego, możliwości przyspieszenia rozwoju gospodarczego.

Na tle przytoczonych faktów z początków rozwoju polskiej informatyki, zrozumiałe jest tak wczesne uruchomienie czasopisma naukowo-technicznego i to w stosunkowo dużym nakładzie 2000 egzemplarzy. I w tym wypadku ujawniła się nasza prężność i wyczucie trendów rozwojowych, ponieważ „Maszyny Matematyczne" były pierwszym czasopismem w krajach RWPG poświęconym wyłącznie technice komputerowej. Nie sposób przy tej okazji nominać zasługi Wydawnictwa Czasopism Technicznych NOT, które jako jedyny z wydawców krajowych, podjęło ryzyko uruchomienia periodyku z dziedziny tak mało wówczas znanej.

Nie będzie przesadą stwierdzenie, że nasze czasopismo, zwłaszcza w początkowym okresie swego istnienia, było ważącym stymulatorem rozwoju polskiej informatyki. Powodowało to oczywiście odpowiednie sprzężenie zwrotne w postaci zwiększenia nakładu i czestotliwości (od 1968 r. miesięcznik) oraz wzrostu zakresu oddziaływania i społecznej rangi czasopisma.

Od 1971 r. nastąpiła zmiana tytułu czasopisma na bardziej komunikatywną INFORMATYKĘ, oraz przyspieszony wzrost nakładu, który na początku 1980 r. osiągnął znaczący, jak na czasopismo naukowo-techniczne, poziom 7400 egzemplarzy.

Mimo powstania wielu specjalizacji informatycznych, stanowiących właściwie odrębne dziedziny wiedzy, a także znacznego zróżnicowania poziomu przygotowania Czytelników, co od wielu już lat wymagało utworzenia co najmniej jednego dodatkowego czasopisma (jedno popularne, drugie naukowo-profesjonalne), INFORMATYKA z konieczności utrzymała w ciągu całego dwudziestolecia swą niezamierzoną monopolistyczną pozycję na krajowym rynku wydawniczym. Powoduje to z jednej strony coraz większe trudności redagowania, ze względu na konieczność wielkiego różnicowania tematów oraz poziomu dostępności artykułów, z drugiej zaś zawartość czasopisma nie może usatysfakcjonować znacznej liczby Czytelników, których zainteresowania znacznie wykraczają poza to, co prezentuje INFORMATYKA. Jakoby na domiar złego, na przestrzeni ostatnich dwóch lat, zamiast nieodzownego już znacznego wzrostu objętości czasopisma, musieliśmy realizować zadanie całkowicie przeciwstawne: zmniejszać objętość wskutek pogłębiającego się deficytu papieru oraz mocy produkcyjnych przemysłu poligraficznego.

Problemy te wystąpiły ze szczególną ostrością w momencie pojawienia się olbrzymiego społecznego zapotrzebowania na publikacje dotyczące mikrokomputerów. Zapotrzebowanie to, pochodzące od wielotysięcznej już rzeszy szczęśliwych posiadaczy tego rodzaju sprzętu, otrzymało od nas natychmiastową odpowiedź w postaci wydzielenia od początku 1984 r. ośmiu stron INFORMATYKI (a więc czwartej części każdego numeru, oczywiście kosztem odpowiedniego ograniczenia pozostałej tematyki) na specjalny dział, nazwany trochę przekornie „mikroKLANEM".

„MikroKLAN" zyskał natychmiast niezwykłą popularność i entuzjastyczne poparcie kilkudziesięciu już tysięcy użytkowników komputerów domowych, co stanowiło dla zespołu redakcyjnego niemałą satysfakcję, a jednocześnie pobudziło do nowych działań.

Oczywiście wysokość nakładu INFORMATYKI i praktyczna niemożliwość jego szybkiego zwiększenia nie dawała żadnych szans zaspokojenia rosnącego z dnia na dzień zapotrzebowania społeczności mikrokomputerowej. W tej sytuacji pod koniec 1984 r. mimo olbrzymich trudności realizacji, rozpoczęliśmy starania o wyodrębnienie mikroKLANU w samodzielną, wysokonakładową publikację. O skali trudności realizacji takiego zadania w dzisiejszych warunkach najlepiej chyba świadczy nie spotykane w naszej 20-letniej działalności, i całkowicie od redakcji niezależne, przeszło 3-miesięczne opóźnienie druku obecnego „jubileuszowego" numeru INFORMATYKI.

Wierzymy, że wspomniane zamierzenia uda się nam w 1986 r. zrealizować. Będzie to niewątpliwie istotny wkład do tak koniecznej dziś masowej edukacji informatycznej społeczeństwa, porównywalny z naszymi działaniami w okresie pionierskim. Dzisiejsze wsparcie informacyjne tematyki mikrokomputerowej to nie tylko zaspokojenie indywidualnych potrzeb wielotysięcznej grupy hobbistów, ale również ważne zadanie społeczne, jakim jest przygotowanie przyszłego użytkownika urządzeń, które już wkrótce wejdą na wyposażenie każdego stanowiska pracy. To także zbliżenie i oswojenie całego społeczeństwa z technologią bliskiego już jutra, której opanowanie i efektywne wykorzystanie zadecyduje o tym, czy utrzymamy się wśród rozwiniętych państw świata.

Spodziewamy się, że podsumowując jubileusz 25-lecia będziemy mogli stwierdzić, że jako czasopismo znacznie lepiej spełniamy nasze obowiązki wobec czytelników, i co ważniejsze, że Polska nie pozostaje w ogonie informatycznego zapóźnienia i odzyskała swą pierwotną rolę prekursora w naszym rejonie Europy.

REDAKCJA

ZBIGNIEW SZKARADNIK
Instytut Informatyki Czasu Rzeczywistego
Politechnika Śląska
Gliwice

Jedynym sposobem nauczenia się nowego języka programowania jest pisanie w nim programów.

B. Kernighan, D. Ritchie [4]

Porównanie języków programowania mikrokomputerów

W środowisku osób mniej lub bardziej profesjonalnie związanych z informatyką trwają dyskusje, który z języków programowania jest najlepszy. Chcąc zweryfikować różne opinie i wyrobić sobie własne zdanie, przetestowałem kilka translatorów popularnych języków programowania.

TESTOWANE TRANSLATORY I PROGRAMY WZORCOWE

Testy przeprowadzono przy użyciu mikrokomputera MERA 60 (odpowiednik LSI-11/03) pod kontrolą systemu operacyjnego RT-60 V04. Testowaniu poddano:

- kompilator języka FORTRAN
- kompilator języka PASCAL
- kompilator języka C
- interpreter języka BASIC
- interpreter języka FORTH

Podstawowe dane wymienionych translatorów zestawiono w tabeli 1.

Do testowania wykorzystuje się zwykle zestaw programów wzorcowych (ang. benchmark program). Za najbardziej reprezentatywny w tej grupie uchodzi program do generacji liczb pierwszych wykorzystujący algorytm Eratostenesa [2]. Oprócz niego wykorzystano algorytm sortowania bąbelkowego.

Wydruki 1—5 przedstawiają teksty źródłowe programu generacji liczb pierwszych w różnych językach programowania. Poszczególne wersje kompilowano przy użyciu wszystkich dostępnych opcji przyspieszających ich realizację.

W przypadku kompilatora FORTRANU wykorzystano możliwość generacji kodu naturalnego (ang. native code, in-line code) oraz kodu kaskadowego (ang. threaded code). Spośród testowanych translatorów jedynie kompilator FORTRANU generuje nadający się od razu do łączenia, relokowalny kod binarny (plik typu OBJ).

Kompilator PASCALA wytwarza program wynikowy w języku asemblera (plik typu MAC), który należy następnie przetłumaczyć przy użyciu systemowego makroasemblera MACRO. Dodatkowo można przed assemblacją dokonać optymalizacji otrzymanego kodu, korzystając z programu o nazwie IMP.

Język C daje programiście możliwość wpływania na czas realizacji programu przez deklarowanie charakteru zmien-

Mgr inż. ZBIGNIEW SZKARADNIK w 1981 r. ukończył studia na Wydziale Automatyki i Informatyki Politechniki Śląskiej (kierunek informatyka). Pracuje jako starszy asystent w Instytucie Informatyki Politechniki Śląskiej, zajmując się oprogramowaniem.

```
{ Eratosthenes sieve prime number program in Pascal }
(*$A-*)
program sieve;
const
  size=8190;
var
  flags:array [0..size] of boolean;
  i, prime, k, count,iter:integer;
begin
  writeln ('10 iterations');
  for iter:=1 to 10 do
    begin
      count:=0;
      for i:=0 to size do
        flags[i]:=true;
      for i:=0 to size do
        if flags[i] then
          begin
            prime:=i+i+3;
            k:=i+prime;
            while k<=size do
              begin
                flags[k]:=false;
                k:=k+prime
              end;
            count:=count+1
          end;
    end;
    writeln (count:4, 'primes')
end {sieve}.
```

Wydruk 1. Program w języku PASCAL

nych (auto, static, register [1]), co zostało wykorzystane do celów porównawczych. Ponadto, kompilator tego języka wytwarza kod wyjściowy w języku asemblera (plik typu

```
100 REM Eratostenes sieve prime number program in Basic
110 DIM F%(8191)
120 PRINT "10 iterations"
130 FOR J%=1 TO 10' C%=0
140 FOR I%=0 TO 8190
150 F% (I%)=1
160 NEXT I%
170 FOR I%=0 TO 8190
180 IF F% (I%)=0 GOTO 260
190 P%=I%+I%+3
200 K%=I%+P%
210 IF K%>8190 GOTO 250
220 F%(K%)=0
230 K%=K%+P%
240 GOTO 210
250 C%=C%+1
260 NEXT I% ' NEXT J%
270 A$="####"
280 PRINT USING A$, C% ;
290 PRINT „ primes"
300 END | REM sieve
```

Wydruk 2. Program w języku BASIC

[1]) Static — zmienne istniejące w pamięci przez cały czas realizacji programu; auto — zmienne istniejące czasowo, tworzone na stosie po wejściu do bloku; register — zmienne rezydujące w rejestrach (por. [4])

.S), wymagający jednakże specjalnego asemblera o nazwie AS.

Wszystkie trzy kompilatory korzystają z systemowego programu łączącego LINK i bibliotek SYSLIB (FORTRAN), PASLIB (PASCAL) i CLIB (C).

```
/* Eratosthenes sieve prime number program in C language */
$$narg = 1;
#define true 1
#define false 0
#define size 8190
char flags [size+1];
main (argc, argv) /* sieve*/
int argc;
char *argv[ ];
  {
  int i,prime,k,count,iter;
  printf ("10 iterations|n");
  for (iter=1;iter<=10;iter++) {
    count=0;
    for (i=0;i<=size;i++)
      flags[i]=true;
    for(i=0;i<=size;i++) {
      if (flags[i]) {
        prime=i+i+3;
        for (k=i+prime;k<=size;k+=prime)
          flags[k]=false;
        count++;
        }
      }
    }
printf ("%d primes|n",count);
} /*sieve */
```

Wydruk 3. Program w języku C

W przypadku języka FORTH kolejne słowa programu są wykonywane natychmiast po rozpoznaniu, oprócz fragmentu umieszczonego między słowami ":" i ";", który jest kompilowany (tłumaczony) na kod kaskadowy (ciąg adresów procedur), wykonywany podczas interpretacji słowa „sieve".

```
( Eratosthenes sieve prime number program in Forth )
8190 constant size
0 variable flags
size allot
: sieve
  ."10 iterations" cr
  10 0 do
    flags size 1 fill
    0 size 0 do
      flags i + c@
      if 1 dup + 3 + dup i +
        begin dup size <
        while 0 over flags + c! over +
        repeat drop drop 1+
      then
    loop
  loop
  . ." primes" cr ; (sieve)
```

Wydruk 4. Program w języku FORTH

Program napisany w języku BASIC nie jest tłumaczony, a jedynie na bieżąco interpretowany i wykonywany. Tak zwana półskompilowana postać programu (plik typu .BAC) jest jedynie oszczędniejszą formą zapisu programów źródłowych na dysku.

```
C     Eratosthenes sieve prime number program in Fortran
      program sieve
      logical*1 flags (8191)
      integer i,j,k,count,iter,prime
      write (5,100)
100   format (x, '10 iterations')
      do 500 iter=1,10
        count=0
        do 200 i=1,8191
          flags i = .true.
200     continue
        do 400 i=1,8191
          if (.not.flags(i)) goto 400
          prime=i+i+1
          count=count+1
          k=i+prime
          if(k.gt.8191) goto 400
          do 300 j=k,8191,prime
            flags(j)=.false.
300       continue
400     continue
500   continue
      write(5,600) count
600   format (x,i4,' primes')
      call exit
      end ! sieve
```

Wydruk 5. Program w języku FORTRAN

WYNIKI TESTOWANIA

W tabelach 2—5 przedstawiono wyniki testowania poszczególnych kompilatorów i interpreterów [2]).

Z zestawienia wynika, że najszybsze programy są generowane przez kompilatory wytwarzające kod naturalny, a nieco wolniejsza jest realizacja kodu kaskadowego. W przypadku języka C stosowanie zmiennych typu auto i static prowadzi do podobnych rezultatów. Przyspieszenie realizacji gwarantuje użycie zmiennych rejestrowych, ale ich liczba jest ograniczona. Wyraźnie najgorszy pod względem czasu realizacji jest interpreter BASICA.

Dla komputerów firmy DEC dostępne są także kompilatory generujące pseudo-kod (ang. p-code), nie było jednak możliwości ich testowania. Można jednak dla pełniejszego porównania opierając się na [2] podać, iż czas realizacji algorytmu Eratostenesa w języku UCSD PASCAL wynosi 221 s, zaś w języku UCSD FORTRAN 281 s na oryginalnej maszynie LSI-11.

[2]) Poszczególne skróty w tabelach 2—5 oznaczają: nat — kod naturalny, thr — kod kaskadowy, reg — kod naturalny z wykorzystaniem zmiennych rejestrowych, src — program źródłowy, blok = 256 słów

Tabela 1. Parametry translatorów

	FORTRAN	BASIC	PASCAL	C	FORTH
Dystrybutor	DEC	DEC	OMSI	DECUS	FIG
Liczba przebiegów	1	—	1	3	—
Wielkość kompilatora (K słów)	51	—	13	25	—
Wielkość asemblera (K słów)	—	—	13	12	—
Wielkość interpretera (K słów)	—	14	—	—	14

Tabela 2. Porównanie czasów realizacji programów

	FORTRAN	FORTRAN	PASCAL	C	C	BASIC	FORTH
Typ generowanego kodu	nat	thr	nat	nat	reg	src	thr
Generowanie liczb pierwszych (s)	34	64	26	24	17	2250	71
Sortowanie bąbelkowe (s)	87	192	122	118	103	11634	314

Należy w tym miejscu podkreślić, że porównywanie translatorów jedynie na podstawie czasu realizacji generowanych programów jest bardzo niepełne. Na osobną ocenę wpływa bowiem jeszcze szereg innych cech, takich jak łatwość uruchamiania i modyfikacji programów napisanych w danym języku, możliwość korzystania z bibliotek, dobra diagnostyka itd.

Jeśli chodzi o pierwszą z tych cech, to wyraźnie najlepszy jest kompilator PASCALA, wyposażony w bardzo dobry symboliczny program uruchomieniowy (debugger) dołączony przy użyciu odpowiedniej opcji kompilatora. Znakomicie ułatwia on uruchomienie programów napisanych w PASCALU i jest wygodniejszy w użyciu niż fortranowy debugger FDT oraz systemowy debugger ODT. Łatwe i szybkie jest uruchamianie programów napisanych w języku BASIC, a po nabyciu pewnego doświadczenia — także w języku FORTH.

```
;       do 100 i=1, 100
        MOV   #2,v$AOTS
        MOV   #1,I
L$AG:   MOV   #3,v$AOTS
;
;       body of loop
;
        INC   I
        CMP   I,#144
        BLE   L$AG
```

Wydruk 6. Rozwinięcie instrukcji 'do' w FORTRANIE

```
; for i := 1 to 100 do
        MOV   %5,—(6)
        MOV   #1,—(6)
        MOV   #144,—(6)
        MOV   2(6),v4(6)
L1:
        CMP   v4(6), (6)
        BGT   L2
;
;       body of loop
;
        INC   v4(6)
        BR    L1
L2:
```

Wydruk 7. Rozwinięcie instrukcji 'for' w PASCALU

```
;       for ( i=1; i<=100; i++ ) ;
        MOV   #1,—10(R5)
        BR    .4
.3:
;
;       body of loop
;
        INC   —10(R5)
.4:
        CMP   —10(R5),#144
        BLE   .3
```

Wydruk 8a. Rozwinięcie instrukcji 'for' w języku C (i-auto)

```
;       for (i =1; i<=100; i++ ) ;
        MOV   #1,i
        BR    .4
.3:
;
;       body of loop
;
        INC   i
.4:
        CMP   i,#144
        BLE   .3
```

Wydruk 8b. Rozwinięcie instrukcji 'for' w języku C (i-static)

```
;       for (i=1; i<=100; i++ ) ;
        MOV   #1,R4
        BR    .4
.3:
;
;       body of loop
;
        INC   R4
.4:
        CMP   R4,#144
        BLE   .3
```

Wydruk 8c. Rozwinięcie instrukcji 'for' w języku C (i-register)

Wielkość generowanego przez kompilator kodu nie jest obecnie aż tak istotna jak niegdyś, ponieważ znacznie wzrosły pojemności pamięci zarówno operacyjnych, jak i zewnętrznych. Ogólnie można powiedzieć, że najmniejsze programy wynikowe zapewnia stosowanie kodów kaskado-

Tabela 3. Porównanie względnych czasów realizacji programów

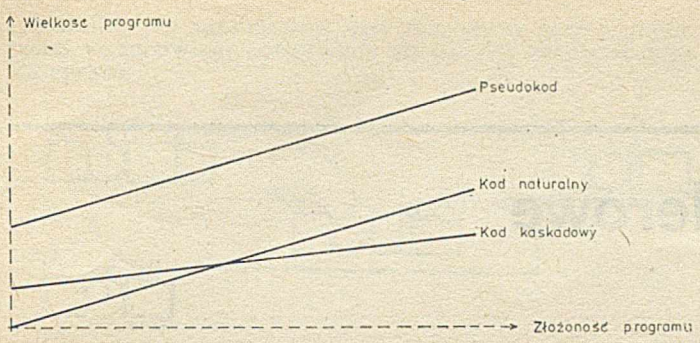
	FORTRAN	FORTRAN	PASCAL	C	C	BASIC	FORTH
Typ generowanego kodu	nat	thr	nat	nat	res	src	thr
Generowanie liczb pierwszych	2.0	3.76	1.53	1.41	1.0	132.3	4.17
Sortowanie bąbelkowe	1.0	2.20	1.40	1.36	1.18	133.7	3.60
Suma czasów względnych	3.0	5.96	2.93	2.77	2.18	266.0	7.77

Tabela 4. Porównanie czasów kompilacji, asemblacji i łączenia

	FORTRAN	FORTRAN	PASCAL	C	C
Typ generowanego kodu	nat	thr	nat	nat	reg
Czas kompilacji (s)	41	36	20	64	64
Czas asemblacji (s)	—	—	50	25	24
Czas łączenia (s)	82	94	38	55	55

Tabela 5. Porównanie wielkości generowanego kodu

	FORTRAN	FORTRAN	PASCAL	C	C	FORTH
Typ generowanego kodu	nat	thr	nat	nat	res	thr
Liczba kompilowanych bajtów	502	430	454	297	247	224
Zajętość pamięci (liczba słów)	7743	8058	8192	6078	6058	4212
Plik wynikowy (liczba słów)	31	32	4	24	24	—



wych. Uwaga ta nie jest jednak słuszna w przypadku programów bardzo małych (rys. 1). Wydaje się, że nadmierna długość generowanego kodu wynika często nie z kiepskiej jakości samego kompilatora, lecz z dołączenia dodatkowych procedur diagnostycznych.

Dla osób pragnących porównać generowany przez kompilatory kod, na wydrukach 6—10 przedstawiono rozwinięcia instrukcji pętli dla języków FORTRAN, PASCAL, C i FORTH.

```
;     101 1 do ... loop
      .WORD   LIT,101.,ONE,XDO
LAB:  .WORD   ; body of loop
      .WORD   XLOOP,LAB-.
```

Wydruk 9. Rozwinięcie instrukcji 'do' w języku FORTH (kod kaskadowy)

```
            .MACRO  NEXT
            MOV     (IP)+,W
            JMP     @(W)+
            .ENDM
;           101 1 do ... loop
            MOV     (IP)+,-(S)
            NEXT
            MOV     (W),-(S)
            NEXT
            MOV     2(S),-(RP)
            MOV     (S),-(RP)
            ADD     #4,S
            NEXT
;
;           body of loop
;
            INC     (RP)
            CMP     (RP),2(RP)
            BGE     1$
            ADD     (IP),IP
            NEXT
1$:         ADD     #4,RP
            ADD     #2,IP
NEXT
```

Wydruk 10. Rozwinięcie kodu kaskadowego dla FORTHA

WNIOSKI KOŃCOWE

Spośród porównywanych języków programowania najwyższą ocenę autora uzyskały PASCAL i C. Należy podkreślić, że na ocenę tę składają się zarówno zalety samego języka (C), jak i zalety kompilatora (PASCAL).

Język C daje programiście wielkie możliwości [4, 7, 9] (może skutecznie zastąpić zarówno asembler, jak i PASCAL), wymaga jednak wyższych niż przeciętne kwalifikacji. Wydaje się, że jest idealnym językiem zwłaszcza dla programistów systemowych, mających dość programowania w języku asemblera.

Testowany kompilator nie jest zbyt wygodny dla osób przyzwyczajonych do korzystania z systemu operacyjnego RT-60 (nie wytwarza wydruków, tablic odwołań zewnętrznych itd.). Jest to konsekwencją rozwiązań przyjętych w systemie operacyjnym UNIX, gdzie funkcje te są realizowane przez osobne programy, dostępne zresztą także w systemie RT. Słabszą stroną testowanego kompilatora jest diagnostyka (automatycznie generowany analizator składni parser).

Zalety PASCALA są powszechnie znane. Testowany kompilator tego języka dopuszcza szereg rozszerzeń w stosunku do języka wzorcowego [6]. W szczególności dozwolona jest segmentacja, bez której trudno wyobrazić sobie sprawne uruchamianie dużych programów. Trochę dokuczliwa jest w PASCALU zbyt natarczywa ochrona programisty przed skutkami błędów, choć jest to logiczna konsekwencja założeń przyjętych w fazie projektowania tego języka [3]. Osobiście uważam, że lepsze byłoby ostrzeganie przed skutkami wykonania jakiejś operacji, zamiast niedopuszczania jej. W niczym nie umniejsza to jednak ogólnej oceny języka, który można polecić zarówno do realizacji poważnych przedsięwzięć, jak i do nauki programowania.

Jeżeli chodzi o FORTRAN, to popularność tego języka maleje, chociaż w niektórych środowiskach (np. w środowisku naukowców) jest on wciąż językiem podstawowym. Testowany kompilator jest produktem dobrym i sprawdzonym [5]. W systemie RT-60 istnieje bogata biblioteka napisanych w FORTRANIE procedur systemowych, matematycznych i statystycznych. Aby móc z tej biblioteki efektywnie korzystać, warto opanować podstawowy w systemie RT język programowania — FORTRAN.

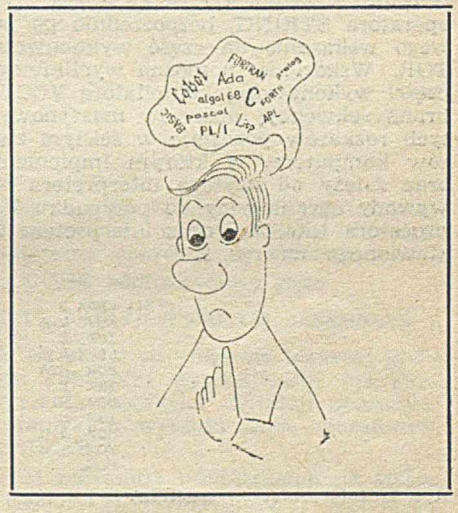
Ostatnio dużo mówi się o języku FORTH [1, 8]. Język ten stawia przed programistą wysokie wymagania, chociażby ze względu na ubogą diagnostykę, bardzo nieczytelny zapis i zastosowaną do zapisu wyrażeń odwrotną notację polską. Jest on właściwie bardziej językiem makroasemblera niż językiem wysokiego poziomu. Z drugiej strony, mechanizm definiowania nowych słów i interakcyjny tryb pracy w połączeniu ze sporą szybkością działania stanowią jego niezaprzeczalne zalety.

Ostatnim z porównywanych języków jest BASIC. Dobrze nadaje się on jedynie do uruchamiania krótkich, podręcznych programów. Przeznaczony jest przede wszystkim dla nieprofesjonalistów i temu zawdzięcza obecnie (i chyba także będzie zawdzięczał w najbliższej przyszłości) swą olbrzymią popularność.

Autor nie miał, niestety, możliwości przetestowania innych języków, które mają do dyspozycji użytkownicy komputerów PDP-11/LSI-11. Oprócz wymienionych mogą oni programować w językach: FORTRAN 77, SIMULA 67, MODULA-2, SNOBOL 4, COBOL, LISP, PROLOG, APL, C++ itd.

LITERATURA

[1] Brodie L.: Starting Forth. Prentice Hall, Englewood Cliffs (NJ), 1981
[2] Gilbreath J., Gilbreath G.: Eratosthenes Revisited. Once more through the Sieve. Byte, Vol. 8, No. 1, January 1983
[3] Jensen K., Wirth N.: Pascal user manual and report. Springer-Verlag, Berlin, 1974
[4] Kernighan W., Ritchie M.: The C programming language. Prentice Hall, Englewood Cliffs (NJ) 1978
[5] Opis języka Fortran-60, Mera-Ster, Katowice, 1981
[6] Opis języka Pascal-60, Mera-Ster, Katowice, 1983
[7] Richtie D. M., Johnson S. C., Lesk M. E., Kernighan B. W.: The C programming language. Dr Dobb's Journal, Number 45
[8] Trojniar W.: Forth — język i system programowania, INFORMATYKA, nr 5, 7, 8, 1984
[9] Winiewski J.: Język programowania C. INFORMATYKA, nr 4—6, 1985.



JAN BIELECKI
Warszawa

Programowanie asemblerowe w języku fig-FORTH

Chociaż język Forth udostępnia środki umożliwiające definiowanie nowych operatorów i instrukcji za pomocą podstawowego zestawu operatorów, często zachodzi potrzeba definiowania operatorów za pomocą sekwencji rozkazów maszynowych.

Podstawy posługiwania się rozkazami maszynowymi

W celu zilustrowania metody postępowania rozpatrzymy na wstępie definicję kompilatora tablic jednowymiarowych. Kompilator taki można zdefiniować za pomocą sekwencji

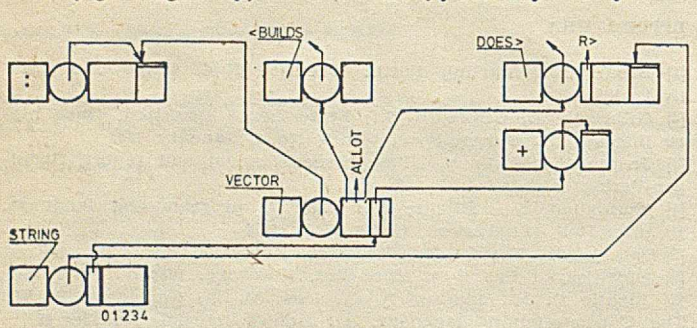
```
: VECTOR <BUILDS ALLOT DOES> + ;
```

Odwołanie się do tej definicji, np. w kontekście

```
5 VECTOR STRING
```

powoduje utworzenie definicji operatora STRING i powiązanie go z innymi operatorami tak, jak to przedstawiono na rys. 1, gdzie:

— pole nazwy wraz z łącznikiem przedstawiono za pomocą kwadratu
— dwubajtowe pole kodu przedstawiono za pomocą okręgu
— pole parametrów przedstawiono za pomocą jednego lub większej liczby prostokątów, poziomą linią wyróżniając te jego fragmenty, które zawierają rozkazy maszynowe.



Rys. 1. Definiowanie słów za pomocą operatorów <BUILDS i DOES>

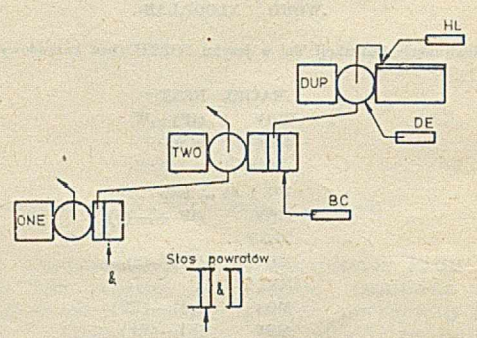
Jak można zauważyć, konsekwencją posłużenia się operatorami <BUILDS i DOES> jest wystąpienie w definicji operatora STRING, bezpośrednio po polu kodu, dwubajtowego wskazania na część wykonawczą kompilatora VECTOR. Wskazanie to można wyeliminować, umieszczając w części wykonawczej kompilatora VECTOR odpowiednio dobraną sekwencję rozkazów maszynowych. Ponieważ dobór tych rozkazów pozostaje w ścisłym związku z listą rozkazów komputera, w którym implementowano język Forth oraz zależy od użytego interpretera wewnętrznego, dalsze wywody ograniczymy do przypadku listy rozkazów mikroprocesora Intel 8080 oraz interpretera wewnętrznego przedstawionego na rys. 2.

```
NEXT: LDAX B
      MOV  L,A
      INX  B
      LDAX B
      MOV  H,A
      INX  B
      MOV  E,M
      INX  H
      MOV  D,M
      XCHG
      PCHL
```

Rys. 2. Interpreter wewnętrzny

W interpreterze tym rejestr BC zawiera wskazanie na następny interpretowany operator, zaś — w chwili podjęcia wykonania operacji wskazanej przez pole kodu bieżącego interpretowanego operatora — rejestr DE zawiera wskazanie na drugi bajt tego pola kodu, a rejestr HL wskazanie na podprogram określony przez to pole.

Spośród trzech wymienionych rejestrów jedynie zawartość BC musi być zachowana między kolejnymi wykonaniami operatorów, co w przypadku gdy bieżący operator jest zdefiniowany za pomocą innych operatorów (a nie za pomocą kodu) wymaga okresowego przechowywania stanu tego rejestru na stosie powrotów.



Rys. 3. Stan rejestrów procesora podczas interpretowania operatorów (błąd w rysunku — znak & powinien być umieszczony nad strzałką)

W celu poglądowego przedstawienia analizowanej dalej implementacji języka Forth na rys. 3 uwidoczniono stan rejestrów BC, DE i HL oraz stan stosu powrotów w chwili podjęcia wykonania operacji DUP, wyrażonej w kodzie maszynowym i wywołanej podczas realizowania operacji TWO, która z kolei została wywołana podczas wykonywania operacji ONE.

Jeśli istotnie dysponujemy mikrokomputerem z listą rozkazów mikroprocesora Intel 8080 oraz zaprezentowaną wyżej implementacją interpretera wewnętrznego, to możemy definicję kompilatora VECTOR przedstawić w postaci

```
: VECTOR CREATE SMUDGE ALLOT
    ;CODE HEX
          E1 C, (POP H)
          13 C, (INX D)
          19 C, (DAD D)
          E5 C, (PUSH H)
          C3 C, NEXT, (JMP NEXT)
          DECIMAL
    SMUDGE
```

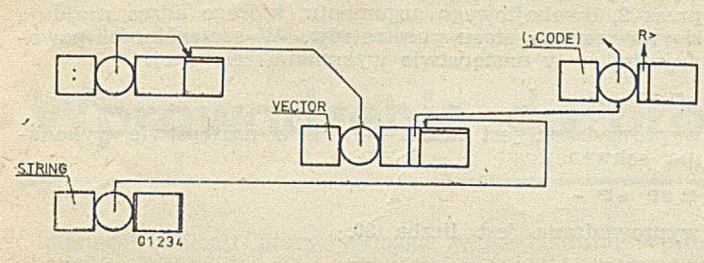
Powyżej założono, że NEXT jest stałą o wartości równej wskazaniu na pierwszy rozkaz interpretera wewnętrznego.

Odwołanie się do tej nowej definicji w rozpatrywanym już kontekście

```
5 VECTOR STRING
```

powoduje jak uprzednio utworzenie definicji operatora STRING i powiązanie go z innymi operatorami, co przedstawiono na rys. 4. Jak łatwo dostrzec, zoptymalizowana definicja operatora STRING jest o dwa bajty krótsza i nie zawiera już pomocniczego wskazania występującego bezpośrednio po polu kodu, tak jak to wystąpiło dla kompilatora VECTOR zdefiniowanego za pomocą słów <BUILDS

i DOES>. To uproszczenie jest rezultatem zdefiniowania części wykonawczej kompilatora za pomocą rozkazów maszynowych.



Rys. 4. Definiowanie słów za pomocą operatora ;CODE

Użycie rozkazów maszynowych w postaci asemblerowej

Mimo iż definicja kompilatora VECTOR, wyrażona za pomocą rozkazów maszynowych, jest w miarę prosta, posługiwanie się szesnastkowymi kodami części operacyjnych tych rozkazów jest w praktyce dość uciążliwe. Pożądane byłoby, aby zamiast operacji poprzedzających komentarze można było użyć wprost napisów występujących w nawiasach.

Takie uproszczenie posługiwania się rozkazami jest możliwe, jednak tradycyjnie rozkazy asemblerowe w języku Forth zapisuje się tak jak inne wyrażenia — w notacji odwrotnej, tj. w rozpatrywanym przypadku w postaci sekwencji

```
D POP
D DAD
H PUSH
NEXT JMP
```

Skoro możliwe jest takie zdefiniowanie np. operatorów H i POP, aby wykonanie sekwencji

```
H POP
```

miało taki sam skutek jak wykonanie sekwencji

```
E1 C,
```

nic nie stoi na przeszkodzie takiemu zredefiniowaniu operatora ;CODE i takiemu zdefiniowaniu operatora C; kompilującego m. in. rozkaz JMP NEXT, aby można zapisać definicję operatora VECTOR w postaci

```
: VECTOR CREATE SMUDGE ALLOT
    ;CODE
        D POP
        D DAD
        H PUSH
    C;
```

gdzie — ;CODE zapoczątkowuje, a C; kończy sekwencję operatorów mających postać zbliżoną do asemblerowego zapisu rozkazów maszynowych.

Posługiwanie się rozkazami maszynowymi nie musi oczywiście ograniczać się do sekwencji definiujących części wykonawcze kompilatorów. Znacznie częściej korzystne jest posłużenie się rozkazami maszynowymi w celu definiowania prostych operatorów przeznaczonych do wykonywania operacji na pojedynczych bitach i bajtach.

Dla uzasadnienia tej tezy rozpatrzymy definicję operatora SWITCH, którego zadaniem jest zamiana miejscami pary bajtów znajdujących się na szczycie stosu parametrów. Mimo iż zadanie wydaje się bardzo łatwe, jego realizacja nie jest krótka i może być wyrażona np. za pomocą definicji

```
: SWITCH DUP SP@ 1+ @ SWAP DROP SWAP DROP ;
```

Znacznie prostszym rozwiązaniem jest posłużenie się rozkazami maszynowymi i wyrażenie definicji operatora SWITCH za pomocą sekwencji

```
CODE SWITCH
    H POP
    A L MOV
    L H MOV
    H A MOV
    H PUSH
C;
```

Jak łatwo obliczyć, pole parametrów drugiego z przytoczonych rozwiązań jest dwukrotnie krótsze, a co więcej, rozwiązanie to jest niemal 10-krotnie szybsze.

Zasady definiowania asemblerów

Uzasadniwszy celowość posługiwania się rozkazami maszynowymi do definiowania prostych operatorów i kompilatorów, pozostaje wykazać, w jaki sposób można standardowy język fig-Forth wzbogacić o podsłownik ASSEMBLER, w którym zostaną zdefiniowane mnemoniki rozkazów asemblerowych oraz podstawowe instrukcje strukturalne.

```
VOCABULARY ASSEMBLER IMMEDIATE
ASSEMBLER DEFINITIONS

HEX

: 8*   DUP + DUP + DUP + ;
: 1MI  <BUILDS C, DOES> C@ C, ;
: 2MI  <BUILDS C, DOES> C@ + C, ;
: 3MI  <BUILDS C, DOES> C@ SWAP 8* + C, ;
: 4MI  <BUILDS C, DOES> C@ C, , ;

0 CONSTANT B
1 CONSTANT C
2 CONSTANT D
3 CONSTANT E
4 CONSTANT H
5 CONSTANT L
6 CONSTANT M
7 CONSTANT A

  E3 1MI XTHL
  37 1MI STC
  1F 1MI RAR
  B0 2MI ORA
  C1 3MI POP
  C5 3MI PUSH
  03 3MI INX
  C3 4MI JMP
  CD 4MI CALL

MOV  SWAP 8* 40 + + C, ;

C2 CONSTANT 0=    ( JNZ )
F2 CONSTANT 0<    ( JP  )

: NOT  8 OR ;
: IF  C, HERE 0 , ;
: THEN  HERE SWAP ! ;

145 CONSTANT NEXT

: CODE  BASE @ CONTEXT @ CREATE [COMPILE] ASSEMBLER ;
: C;  NEXT C3 ( JMP NEXT ) C, , CONTEXT ! BASE ! SMUDGE ;

DECIMAL
```

Rys. 5. Definicje podsłownika ASSEMBLER

Aby nie czynić opisu nadmiernie rozwlekłym i ograniczyć się jedynie do podania zasad definiowania asemblerów, na rys. 5 przytoczono niezbędne definicje umożliwiające posługiwanie się mnemonikami użytych dalej rozkazów i symbolicznymi oznaczeniami rejestrów oraz zdefiniowano kilka przykładowych operatorów i instrukcji umożliwiających asemblerowe programowanie strukturalne.

Wszystko to łącznie stwarza zespół środków umożliwiających przedstawienie definicji operatora ?SWAP, zdefiniowanego bez użycia rozkazów maszynowych jako

```
: ?SWAP IF SWAP THEN ;
```

w jego alternatywnej postaci

```
CODE ?SWAP
    H POP
    A L MOV
    H ORA
    0= NOT IF
      H POP
      XTHL
      H PUSH
    THEN
C;
```

Aby wyjaśnić, w jaki sposób wykonanie przytoczonej sekwencji operatorów prowadzi do utworzenia definicji operacji ?SWAP, rozpatrzymy pokrótce jej elementy:

- wykonanie operacji CODE zdefiniowanej jako

```
: CODE BASE @ CONTEXT @ CREATE [COMPILE] ASSEMBLER ;
```

powoduje zapamiętanie na stosie parametrów bieżącej podstawy liczb i bieżącego kontekstu wyszukiwania operatorów, a następnie utworzenie nagłówka definicji operatora ?SWAP i ustawienie kontekstu wyszukiwania operatorów na podstawnik ASSEMBLER,

- wykonanie operacji H powoduje umieszczenie na stosie parametrów stałej o wartości 4, identyfikującej rejestr H,

● wykonanie operacji POP skompilowanej za pomocą kompilatora 3MI powoduje wykonanie sekwencji

C ₪ SWAP 8* + C,

tj. utworzenie na stosie parametrów, a następnie przeniesienie do słownika bajtu reprezentującego rozkaz

POP H,

● wykonanie sekwencji operacji

A L MOV

powoduje w analogiczny sposób umieszczenie w słowniku bajtu reprezentującego rozkaz

MOV A,L

● wykonanie sekwencji operacji

0= NOT IF

powoduje umieszczenie w słowniku rozkazu skoku JZ z adresem rozkazu

JMP NEXT

umieszczonego w słowniku w ramach wykonania operacji C; (przyjęto, że początek pętli interpretera wewnętrznego znajduje się pod adresem 145)

● wykonanie operacji C; zdefiniowanej jako

: C; NEXT C3 C, , CONTEXT ! BASE ! SMUDGE ;

powoduje umieszczenie w definicji operatora ?SWAP rozkazu maszynowego JMP NEXT, przywrócenie ze stosu parametrów kontekstu wyszukiwania operatorów i podstawy liczb, a ponadto usunięcie z nagłówka definiowanego operatora bitu „smudge" umieszczonego tam przez CREATE.

● ostatecznie pole parametrów operatora ?SWAP uzyskuje postać ciągu rozkazów, który można przedstawić symbolicznie jako

```
   POP H
   MOV A,L
   ORA H
   JZ skip
   POP H
   XTHL
   PUSH H
skip:
   JMP NEXT
```

Środki do programowania hybrydowego Forth-asembler

Posługiwanie się operatorami CODE, ;CODE i C; umożliwia definiowanie części wykonawczych kompilatorów oraz definiowanie prostych operatorów za pomocą rozkazów maszynowych. Dysponując takimi środkami naturalne wydaje się pytanie, czy możliwe jest zdefiniowanie w języku takich dodatkowych operatorów, aby w dowolnym miejscu definicji można było posługiwać się na przemian operatorami i rozkazami maszynowymi.

Chociaż trudno jest ocenić zakres zastosowań takich operatorów występujących w kontekście

operatory ASM (rozkazy) FORTH operatory

oraz

rozkazy FORTH (operatory) ASM rozkazy

przeanalizowanie ich definicji należy zdaniem autora uznać za nietrywialne ćwiczenie praktyczne stanowiące sprawdzian biegłości posługiwania się językiem Forth.

```
: 2/  DUP @
      ASM( H POP
           A H MOV
           A ORA
           0< IF
             STC
           THEN
           RAR
           H A MOV
           A L MOV
           RAR
           L A MOV
           H PUSH )FORTH
      SWAP ! ;
```

Rys. 6. Użycie operatorów ASM (i) FORTH do zdefiniowania operatora 2/

```
CODE 2*  H POP
         H PUSH
         H PUSH
         FORTH( @ DUP + )ASM
         D POP
         H POP
         M E MOV
         H INX
         M D MOV
         C;
```

Rys. 7. Użycie operatorów FORTH (i) ASM do zdefiniowania operatora 2*

Jako przygotowanie do tego zadania na rysunkach 6 i 7 przedstawiono hybrydowe definicje operatorów 2/ i 2*. Pierwszy z nich służy do dzielenia, a drugi do mnożenia przez 2, dwubajtowego argumentu, którego adres znajduje się na szczycie stosu parametrów. W szczególności powoduje to, że w następstwie wykonania sekwencji

—81 SP ₪ 2/ .

wyprowadzana jest liczba —41, a w następstwie wykonania sekwencji

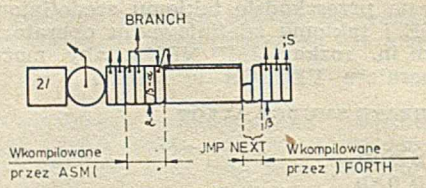
40 SP ₪ 2*

wyprowadzana jest liczba 80.

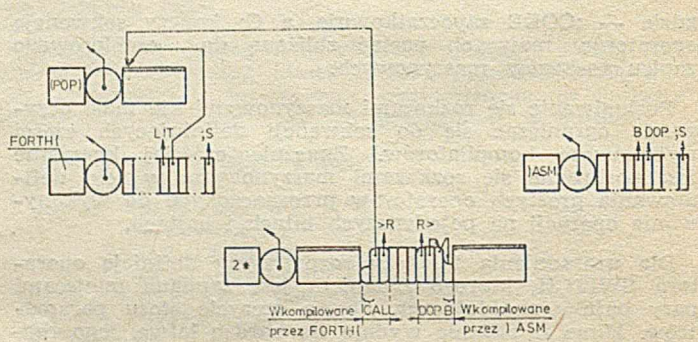
```
FORTH DEFINITIONS
: ASM(  BASE @ CONTEXT @ HERE 6 + ,
        COMPILE BRANCH HERE 0 ,
        HERE 2+ , [COMPILE] ASSEMBLER
        [COMPILE] C ; IMMEDIATE

ASSEMBLER DEFINITIONS
: )FORTH  NEXT JMP HERE OVER - SWAP !
          CONTEXT ! BASE ! ] ;


ASSEMBLER DEFINITIONS
CODE (POP)  L C MOV
            H B MOV
            XTHL
            C L MOV
            B H MOV
            C;
: FORTH(  BASE @ CONTEXT @ ' (POP),
          CALL ] COMPILE )R ;

FORTH DEFINITIONS
: )ASM  COMPILE R) HERE 2+ , HERE 2+ ,
        ASSEMBLER B POP FORTH
        CONTEXT ! BASE ! [COMPILE] ASSEMBLER
        [COMPILE] C ; IMMEDIATE
```

Rys. 8. Definicje operatorów do programowania hybrydowego

Proponowaną implementację czterech nowych operatorów języka fig-Forth, umożliwiających programowanie hybrydowe przedstawiono na rys. 8. Dla ułatwienia ich analizy na rys. 9 przytoczono definicję operatora 2/, a na rys. 10 wybrane fragmenty definicji operatorów FORTH(, ASM, (POP) i 2*.



Rys. 9. Definicja operatora 2/



Rys. 10. Fragmenty definicji

LITERATURA

[1] Bielecki J.: Definiowanie kompilatorów i instrukcji strukturalnych w języku fig-Forth. Informatyka, nr 7, 1985

[2] Brodie L.: Starting Forth. Prentice Hall, Englewoods Cliffs, New Jersey, 1981

[3] McCabe C.: Forth Fundamentals. Dilithium Press, Beaverton, Oregon, 1983

[4] Thomasson D.: Advanced Spectrum Forth. Melbourne House, South Melbourne, 1984

[5] Trojniar W.: Forth — język i system programowania. Informatyka, nr 5, 7, 8, 1984

STANISŁAW GASIK
Uniwersytet Warszawski

Wprowadzenie do programometrii

Jakiego nakładu pracy wymaga wyprodukowanie programu? Czy trudno jest zrozumieć działanie programu? Do jakiego stopnia przejrzysta jest dokumentacja programu? Od pewnego czasu w literaturze informatycznej pojawiają się próby odpowiedzi na podobne pytania. W publikacjach takich podaje się konkretne cechy programów, wpływające na ich złożoność psychologiczną oraz metody obliczania miar tych cech.

Podstawowym celem tego artykułu jest uporządkowanie metod określania miar złożoności. Wyodrębniono w nim grupy czynników, które należy uwzględniać przy definiowaniu miar złożoności oprogramowania, pokazano strukturę zależności między nimi, wskazano dziedziny nauki, które powinny być wykorzystywane przy określeniu miar złożoności. Wreszcie podano definicję programometrii, a w zakończeniu określono miejsce tej nowej dziedziny w informatyce oraz jej perspektywiczne cele.

PODSTAWOWE CZYNNIKI ZŁOŻONOŚCI OPROGRAMOWANIA

Najłatwiej jest zrozumieć działanie programu nie zawierającego żadnej instrukcji (o ile dany język programowania takie programy dopuszcza). Im dłuższy jest program, tym trudniej prześledzić jego działanie. Oznacza to, że jednym z podstawowych czynników złożoności oprogramowania jest r o z m i a r. Przekonanie, że rozmiar jest istotnym czynnikiem złożoności oprogramowania można znaleźć w pracy Halsteada [7]. Halstead określa takie wartości, jak nakład pracy potrzebnej do wyprodukowania programu czy przewidywaną liczbę błędów jako funkcje liczby użytych operatorów i argumentów.

Nie zawsze dwa programy mające taki sam rozmiar, są tak samo złożone. Ogólnie wiadomo, że programy zawierające dużą liczbę skoków, instrukcji warunkowych i innych instrukcji sterujących, są trudne do zrozumienia [13]. Z drugiej strony, im bardziej — mówiąc nieformalnie — struktura systemu jest podobna do struktury drzewiastej, tym mniej skomplikowany jest system [5]. Oznacza to, że nie tylko rozmiar, ale i s t r u k t u r a jest czynnikiem złożoności oprogramowania.

Zwróćmy uwagę, że czym innym jest struktura pojedynczego modułu, a czym innym — struktura systemu programowego. Strukturę systemu wyznacza sposób oddziaływania na siebie poszczególnych składowych — modułów. Przez strukturę modułu rozumie się sposób przepływu sterowania przez ten moduł. Można także mówić o strukturze jeszcze niższego rzędu, tj. strukturze sekwencyjnie wykonywanych fragmentów modułów — bloków podstawowych. Za strukturę bloku podstawowego można uważać występujące w nim zależności pomiędzy (czynnymi i biernymi) odwołaniami do danych.

Przykładem miary struktury systemu jest miara złożoności strukturalnej [5]

C — S = liczba_odwołań_do_modułów / liczba_modułów

Miara ta (przy ustalonej liczbie modułów) przyjmuje najmniejszą wartość, gdy najmniejsza jest liczba odwołań do modułów, czyli w przypadku struktury drzewiastej.

Inną miarą struktury systemu jest miara modularności [10]

$$U = \sum_{I=0}^{N} IF(I) \Big/ \sum_{I=0}^{N} EF(I)$$

gdzie I oznacza poziom w hierarchii systemu (0 — korzeń, 1 — moduły wywoływane bezpośrednio przez korzeń, itd.), IF(I) jest liczbą funkcji importowanych na poziomie I, EF(I) — liczbą funkcji eksportowanych na poziomie I.

Miara ta jest prostym uogólnieniem miary złożoności strukturalnej — dopuszcza wiele funkcji w jednym module.

Przykładami miar odnoszących się do struktury modułów są: liczba cyklomatyczna [11], liczba węzłów [16], miara względnej złożoności logicznej [5]. Wiele innych miar tego rodzaju opisano w zbiorczej pracy Hockera i współautorów [8].

Liczba cyklomatyczna jest to liczba warunków logicznych, zwiększona o 1.

Żeby określić liczbę węzłów, należy na tabulogramie narysować wszystkie skoki w postaci wektorów — od instrukcji GOTO do etykiety (autorzy definiują swoją miarę dla programów fortranowskich) i policzyć punkty ich przecięć — węzły.

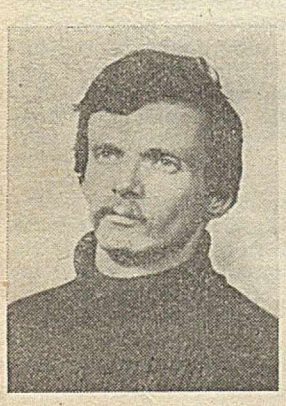
Miara względnej złożoności logicznej ma następującą postać: C — 1 = liczba_binarnych_instrukcji_decyzyjnych / liczba_wszystkich_instrukcji.

Miarą złożoności bloku podstawowego może być liczba instrukcji, w których nadaje się wartości, wykorzystywane przez pewne zmienne w danym bloku. Liczba ta jest podstawą miary złożoności [12].

Żeby poznać kolejne czynniki wpływające na złożoność oprogramowania, rozważmy dwa programy o tym samym rozmiarze i takiej samej strukturze. Okazuje się, że wcale nie muszą one być tak samo zrozumiałe, co ilustrują poniższe przykłady programów napisanych w PL/I.

```
/*Przykład 1*/

IF A1 > A2 THEN
    DO; /*WYKONAJ, JEŚLI A1 JEST WIĘKSZE OD A2*/
        CALL A3 (A4);
        WRITE FILE (A5) FROM (A1);
        A6 = A6 + 1;
    END;

/*Przykład 2*/

IF TEMPERATURA > GRANICA THEN
    DO;
        CALL WŁĄCZ (CHŁODZENIE);
        WRITE FILE (ALARM) FROM (TEMPERATURA);
        PRZEKROCZENIE = PRZEKROCZENIE + 1;
    END;
```

W przykładzie 1 trudno zrozumieć, jaką funkcję wykonuje ten fragment programu. Powodem jest niewłaściwy dobór nazw oraz bezsensowny „składniowy" komentarz.



Mgr STANISŁAW GASIK ukończył studia w 1976 r. na Wydziale Matematyki i Mechaniki Uniwersytetu Warszawskiego. Pracował w Instytucie Maszyn Matematycznych i w Politechnice Warszawskiej. Obecnie jest kierownikiem Ośrodka Komputerowego Wydziału Psychologii Uniwersytetu Warszawskiego. Był wykładowcą PL/I programowania strukturalnego i technologii programowania w Centrum Szkolenia Informatycznego ZETO Łódź, Centrum Projektowania i Zastosowań Informatyki w Warszawie i na kursach zastosowań matematyki, organizowanych przez PAN i Uniwersytet Warszawski. Zainteresowania zawodowe obejmują inżynierię i psychologię programowania oraz metryki oprogramowania.

Natomiast nazwy użyte w przykładzie 2 jednoznacznie opisują wykonywaną funkcję.

Przykłady te pokazują, że istnieje następna grupa cech, określających złożoność programów. Ten zespół cech będziemy nazywać sugestywnością. Jest to stopień korelacji brzmienia nazw, komentarzy i stałych użytych w programie, z ich znaczeniem.

Przykładem (niekompletnej) miary sugestywności programu może być średnia długość zmiennych [3]. Cechy programów związane z sugestywnością nie doczekały się zbyt wielu miar.

Można jeszcze bardziej zwiększyć złożoność fragmentu programu z przykładu 1, nie zmieniając przy tym jego rozmiaru, struktury ani sugestywności.

```
/*Przykład 3*/

IF A1 > A2 THEN DO; /*WYKONAJ, JEŚLI A1 JEST

    WIĘKSZE OD A2*/
CALL A3 (A4); WRITE FILE (A5) FROM (A1); END;
```

Przykład ten jest niewątpliwie jeszcze mniej czytelny niż przykład 1. Oznacza to, że zmieniliśmy wartość kolejnego czynnika złożoności programu. Tym czynnikiem jest forma. Forma jest to graficzny układ programu na nośnikach informacji. Elementami formy są nie znaczące spacje, akapity, puste wiersze, rozmieszczenie na stronach itp. Wyniki prób mierzenia formy programu są zawarte w pracy Joergensena [9]. W celu określenia tej miary programu autor wyodrębnia takie wielkosci, jak:

- łączna liczba spacji w lewych marginesach,
- łączna liczba pustych wierszy.

Literatura dotycząca mierzenia formy programu jest również dość uboga, a zaprezentowanych prób nie można w pełni uznać za udane.

Powyżej wyodrębniono cztery podstawowe czynniki złożoności oprogramowania: rozmiar, strukturę, sugestywność i formę. Prezentując miary struktury pokazano, że inne miary odnoszą się do struktury systemu, inne do struktury modułu, a jeszcze inne do struktury bloku podstawowego. Analogicznie dla pozostałych czynników — rozmiaru, sugestywności i formy — powinno się określić różne miary, w zależności od poziomu obiektów, do których miara ma się odnosić. Oczywiste jest, na przykład, że co innego mamy na myśli mówiąc o przejrzystej formie zapisu systemu, co innego mówiąc o przejrzystej formie zapisu modułu (akapity instrukcji warunkowych, pętli itp.), a co innego — mówiąc o formie bloku podstawowego (np. wyodrębnienie poszczególnych operacji arytmetycznych).

Żeby mówić o różnych miarach poszczególnych czynników w zależności od miejsca obiektu w naturalnej hierarchii (system, moduł, blok podstawowy), należy wprowadzić kilka definicji.

System jest to zbiór współdziałających ze sobą modułów.

Moduł jest to niezależnie tłumaczona jednostka programowa, np. podprogram w FORTRANIE, procedura zewnętrzna w PL/I, czy pakiet w ADZIE.

Blok podstawowy [1] jest to ciąg instrukcji, nie zawierających rozgałęzień i nie mający (oprócz, być może, początku) punktów mogących być celami skoków. Jeśli sterowanie osiągnie początek bloku podstawowego, to musi dojść do jego końca. Blok podstawowy czasami bywa nazywany **sekwencją logiczną** [14].

Obiekt bezpośrednio niższego poziomu (dla systemu — moduł, dla modułu — blok podstawowy), wykorzystywany przez dany obiekt, będziemy nazywali **obiektem bezpośrednio podporządkowanym** danemu.

Obiekty dowolnych niższych poziomów, wykorzystywane (pośrednio lub bezpośrednio) przez dany obiekt, będziemy nazywali **obiektami podporządkowanymi** danemu. Ponieważ wyodrębniono trzy poziomy obiektów, więc obiektami nie bezpośrednio podporządkowanymi mogą być tylko bloki podstawowe dla systemu.

System odpowiada poziomowi procesów, moduł — poziomowi sterowania, zaś blok podstawowy — poziomowi obliczeń. Gilb [5] nazywa poziom modułu poziomem **logicware**, zaś poziom bloków podstawowych poziomem **dataware**.

W dalszym ciągu artykułu słowo **obiekt** będzie oznaczało system, moduł lub blok podstawowy.

ZŁOŻONOŚĆ OPERACYJNA

Poniżej omówiono zależności między wyodrębnionymi czynnikami złożoności oraz wpływ tych zależności na miary złożoności oprogramowania.

Wyróżnione czynniki złożoności oprogramowania są istotne wtedy, gdy chcemy ocenić nakład pracy potrzebnej do zrozumienia jak program wykonuje swoje funkcje. Czynniki te tworzą dziedzinę złożoności operacyjnej oprogramowania.

W dotychczasowych publikacjach traktujących o złożoności operacyjnej często zajmowano się jednym z czynników złożoności, zaniedbując pozostałe. Takie podejście dominuje w cytowanej pracy Halsteada [7], która dotyczy wpływu rozmiaru na złożoność oprogramowania. Analogicznie Mc Cabe w pracy [11] zajmuje się tylko wpływem struktury na złożoność.

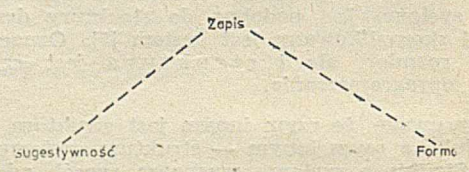
Istnieją publikacje, których autorzy próbują wyliczać pewne miary podstawowych czynników złożoności, a następnie — przy użyciu elementarnych operacji arytmetycznych — tworzyć jedną miarę złożoności. Najbardziej przejrzystą miarą tego rodzaju wydaje się być tzw. miara czytelności (zwróćmy uwagę na brak systematyki w nazywaniu pojęć związanych ze złożonością oprogramowania) De Younga i Kampena [3]. Jako wskaźnik czytelności określają oni:

$$R = 0{\cdot}295 \cdot VAR - 0{\cdot}499 \cdot NSL + 0{\cdot}13 \cdot CYCLO$$

gdzie VAR — oznacza średnią długość zmiennych, NSL — liczbę linii zawierających instrukcję, a CYCLO — liczbę cyklomatyczną Mc Cabe.

Zdaniem autora właściwą miarę złożoności oprogramowania należy określać przez wyodrębnienie elementarnych czynników złożoności oraz ustalenie zależności między nimi. Poniżej zajmiemy się zależnościami między podstawowymi czynnikami złożoności oprogramowania.

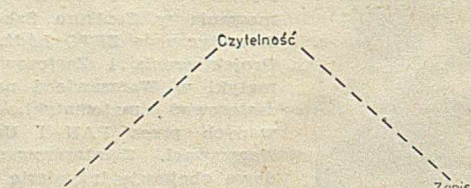
Sugestywność i forma są czynnikami, które w zasadzie nie wpływają na semantykę programów. Jeśli język programowania jest niepozycyjny, to zdania można dowolnie rozmieszczać na nośnikach informacji. Podobnie komentarze i brzmienie nazw nie wpływają na działanie programu. W związku z tym uzasadnione jest wyodrębnienie niepodstawowego czynnika złożoności, który nazwiemy zapisem programu. Zapis programu jest określany przez sugestywność i formę. Strukturę tego czynnika można zobrazować schematem (rys. 1):



Rys. 1. Czujniki określające zapis

Z zależności tych wynika, że miara jakości zapisu powinna być zależna od miar sugestywności i formy programu.

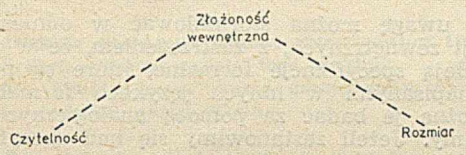
Zapis jest czynnikiem określającym, jak przejrzyście przedstawiono strukturę programu. Te dwa czynniki — struktura i zapis — wyznaczają szybkość rozumienia programu w trakcie jego lektury. Wprowadzimy więc następny, niepodstawowy czynnik złożoności, nazywany czytelnością. Czytelność jest to średni czas, potrzebny do zrozumienia podstawowej jednostki programu. Za podstawową jednostkę rozmiaru można przyjąć np. zdanie lub jednostkę leksykalną (zmienna, stała, operator itd.). Czytelność programu można przedstawić za pomocą następującego schematu (rys. 2):



Rys. 2. Składowe czytelności

Miary czytelności powinny uwzględniać cechy związane ze strukturą i z zapisem programu.

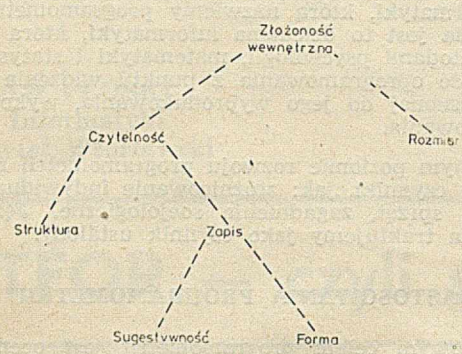
Jeśli udaje się określić tak rozumianą miarę czytelności, a ponadto znany jest rozmiar programu, to można obliczyć czas potrzebny do zrozumienia działania całego programu, będący miarą złożoności wewnętrznej programu (rys. 3).



Rys. 3. Złożoność wewnętrzna

Tak więc miary złożoności wewnętrznej powinny być uzależnione od miar czytelności i rozmiaru.

Po uwzględnieniu miar czytelności i zapisu, otrzymamy następujący schemat złożoności wewnętrznej oprogramowania (rys. 4).



Rys. 4. Pełna struktura złożoności wewnętrznej

Każda miara, określająca złożoność wewnętrzną obiektu (dowolnego poziomu), powinna odzwierciedlać wskazane wyżej zależności pomiędzy czynnikami elementarnymi.

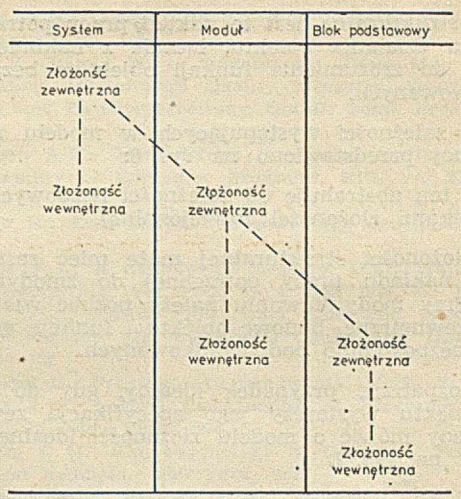
ZŁOŻONOŚĆ FUNKCJONALNA

Rozważania powyższe dotyczyły czynników związanych ze złożonością operacyjną oprogramowania oraz zależności między czynnikami określającymi nakład pracy potrzebnej do zrozumienia jak program wykonuje swoje funkcje. Poniżej omówiono czynniki określające nakład pracy potrzebnej do zrozumienia co program wykonuje. Czynniki te tworzą dziedzinę złożoności funkcjonalnej oprogramowania. Podobnie jak w dziedzinie złożoności operacyjnej, podamy teraz zależności pomiędzy czynnikami złożoności funkcjonalnej. Są one wyznaczone przez zależności między miarami tych czynników.

Aby poznać funkcje obiektu wyłącznie na podstawie jego kodu, należy zapoznać się z całością jego kodu i kodu obiektów podporządkowanych. Nakład pracy potrzebny do zrozumienia funkcji obiektu nazwiemy złożonością zewnętrzną. Model złożoności funkcjonalnej, w którym dla zrozumienia funkcji obiektu trzeba zrozumieć całość kodu zarówno tego obiektu, jak i obiektów podporządkowanych, nazwiemy modelem złożoności totalnej. Model ten schematycznie przedstawiono na rys. 5.

Trudno jednak wyobrazić sobie poznanie funkcji dużych systemów wyłącznie na podstawie lektury ich kodu. Dlatego też model złożoności totalnej — w szczególności na poziomie systemu — jest przeznaczony do innych celów.

Van Tassel [13] twierdzi, że 50—90% pracy programistów przypada na uruchamianie programów. Łatwo uruchamia się te programy, które są (łatwo) zrozumiałe. Wynika stąd, że nakład pracy potrzebnej do wyprodukowania obiektu jest proporcjonalny do nakładu pracy potrzebnej do zrozumienia jego działania. Taką opinię podzielają m. in. Halstead [7] i Gordon [6]. Model złożoności totalnej może zatem być przydatny do oceny nakładu pracy potrzebnej do wyprodukowania obiektu. Stwierdzenie to odnosi się do obiektów dowolnego poziomu.

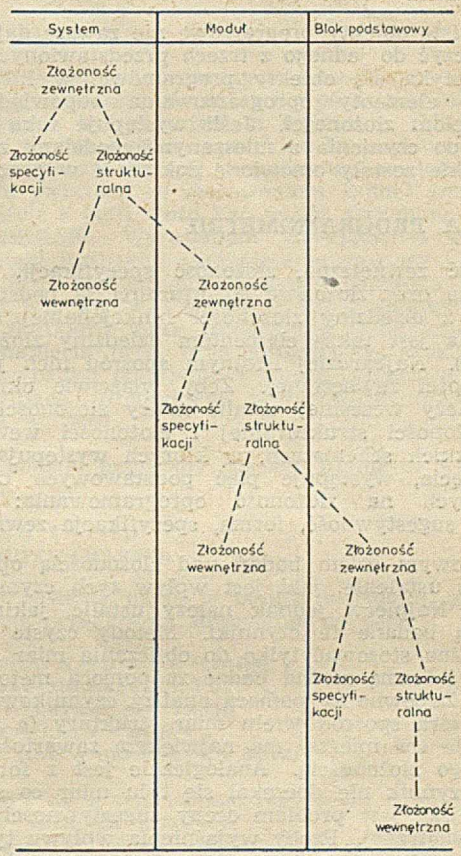


Rys. 5. Model złożoności totalnej

W praktyce większość obiektów programowych ma swoje specyfikacje zewnętrzne. Specyfikacja zewnętrzna jest to dowolny opis funkcji obiektu programowego — w postaci formalnej lub nieformalnej. Może to być opis translatora czy systemu aplikacyjnego w formie książkowej. Za specyfikację zewnętrzną można także uważać komentarze w nagłówku podprogramu, określające — mniej lub bardziej dokładnie — jego funkcję.

Nader często zdarza się, że specyfikacje zewnętrzne obiektów programowych są nie dość dobrze napisane, np. nie zawierają wszystkich potrzebnych informacji. Takie specyfikacje mogą co najwyżej ułatwić poznanie funkcji oprogramowania. Żeby zrozumieć dokładnie, co program wykonuje, trzeba poznać także jego budowę i funkcje wykonywane w nim przez obiekty bezpośrednio podporządkowane.

Składowe te będą elementami modelu złożoności strukturalnej. Do przedstawienia schematu tego modelu potrzebne są jeszcze dwa nowe pojęcia:



Rys. 6. Model złożoności strukturalnej

Złożoność specyfikacji jest to nakład pracy potrzebny do zrozumienia specyfikacji zewnętrznej obiektu.

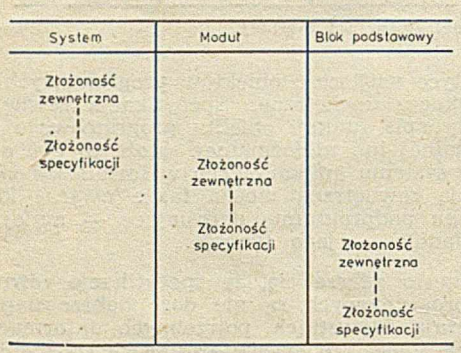
Złożoność strukturalna jest to nakład pracy potrzebnej do zrozumienia działania obiektu, łącznie z nakładem pracy potrzebnej do zrozumienia funkcji obiektów bezpośrednio podporządkowanych.

Schemat zależności występujących w modelu złożoności strukturalnej przedstawiono na rys. 6:

Schemat ten abstrahuje od zależności liczbowych pomiędzy czynnikami złożoności funkcjonalnej.

Model złożoności strukturalnej może mieć zastosowanie do oceny nakładu pracy potrzebnej do zmodyfikowania obiektu. Przy modyfikowaniu należy poznać właśnie specyfikacje zewnętrzne, budowę obiektu i funkcję wszystkich obiektów bezpośrednio podporządkowanych.

Warto rozpatrzyć przypadek idealny, gdy do poznania funkcji obiektu wystarczy jego specyfikacja zewnętrzna. Można wtedy mówić o modelu złożoności idealnej, przedstawionym na rys. 7.



Rys. 7. Model złożoności idealnej

Model złożoności idealnej może mieć zastosowanie tylko do oceny nakładu pracy potrzebnej do zrozumienia funkcji obiektu. Wielkości występujące w nim są niewystarczające do oceny nakładu pracy potrzebnej do modyfikowania obiektu — brak tu bowiem jakichkolwiek informacji o budowie obiektu.

Rzeczywiste obiekty programowe nie zawsze dają się tak łatwo zaliczyć do jednego z trzech przedstawionych modeli. Często spotyka się obiekty programowe, w których poszczególne elementy oprogramowania odpowiadają różnym modelom złożoności. Jeśli występuje taka sytuacja, to mamy do czynienia z mieszanymi modelami złożoności. Modele takie zostały omówione dokładnie w pracy [4].

DEFINICJA PROGRAMOMETRII

Złożoność zewnętrzna, złożoność specyfikacji, złożoność wewnętrzna oraz złożoność strukturalna są podstawowymi pojęciami z dziedziny złożoności funkcjonalnej (złożoność wewnętrzna jest także elementem dziedziny złożoności operacyjnej). Najbardziej istotnym spośród nich jest pojęcie złożoności zewnętrznej. Żeby właściwie określić jej miarę, należy wcześniej ustalić miary złożoności specyfikacji, złożoności strukturalnej i złożoności wewnętrznej. We wszystkich schematach, w których występują wspomniane pojęcia, występuje pięć podstawowych czynników wpływających na złożoność oprogramowania: rozmiar, struktura, sugestywność, forma, specyfikacja zewnętrzna.

Podstawowym celem badań nad złożonością oprogramowania jest ustalenie, jaki jest wpływ tych czynników na złożoność. Najpierw jednak należy ustalić, jakimi metodami będą badane te czynniki. Metody czysto matematyczne można stosować tylko do obliczania miar rozmiaru. Strukturę i formę można badać za pomocą metod statystycznych. Zapewne za pomocą analizy czynnikowej można wykryć, która spośród wielu miar struktury (a wszystkie rzeczywiście coś mierzą) ma największą zawartość czynnika, zwanego złożonością. Analogicznie jest z formą, chociaż ten czynnik nie doczekał się tylu miar co struktura. Najtrudniejszy jest problem oceny sugestywności i specyfikacji zewnętrznej. Próby wyjaśnienia wpływu tych czynników na złożoność za pomocą metod czysto ilościowych trudno uznać za udane. Dość łatwo jest bowiem skonstruować przykład, w którym dowolnie długi komentarz będzie bardzo zły. Podobnie ani fizyczne zróżnicowanie, ani długość nazwy nie muszą niczego wyjaśniać. Dopiero dodanie czynnika psychologicznego — np. motywacji — może pomóc częściowo rozwiązać problem. Ostatecznie, dopiero metody semantyczne języka naturalnego (co znaczy nazwa zmiennej „duży"), być może w powiązaniu z metodami sztucznej inteligencji, pozwolą właściwie określać jakość czynników, które nazywamy sugestywnością.

Podobną uwagę można sformułować w odniesieniu do specyfikacji zewnętrznych, z zastrzeżeniem, że w tej dziedzinie istnieją specyfikacje formalne, które są po prostu tekstami napisanymi w innych językach formalnych, i dlatego można je badać za pomocą analogicznych metod, jak programy. Jeżeli zastanowimy się nad specyfikacjami formalnymi, to dojdziemy do wniosku, że ich autorzy często zaniedbują wszystkie albo większość czynników złożoności oprogramowania. Najwięcej kłopotów mają oni z nadaniem specyfikacji rozsądnych rozmiarów. Pamiętajmy, że pisząc specyfikacje formalne, powinniśmy zwracać uwagę na wszystkie podstawowe czynniki złożoności. Zaś jednym z podstawowych zadań twórców przyszłych metod specyfikacji powinno być dostosowanie ich do możliwości ludzkich.

Przeprowadzone rozważania prowadzą do definicji dziedziny informatyki, którą nazwiemy programometrią. **Programometria** jest to dziedzina informatyki, która posługując się metodami psychologii, matematyki i statystyki, bada złożoność oprogramowania z punktu widzenia nakładu pracy potrzebnej do jego wyprodukowania, wykorzystania i modyfikowania.

Na obecnym poziomie rozwoju programometrii zaniedbujemy takie czynniki, jak: zróżnicowanie indywidualne programistów, sprzęt, zagadnienia socjologiczne. Język programowania traktujemy jako czynnik ustalony.

CELE I ZASTOSOWANIA PROGRAMOMETRII

Podstawowym celem programometrii jest ocena złożoności oprogramowania en block. Do realizacji tego celu prowadzą dwie drogi. Pierwsze rozwiązanie polega na podaniu miar pięciu czynników podstawowych dla każdego obiektu każdej kategorii i traktowaniu tego zbioru liczb jako zbioru wskaźników. Drugie rozwiązanie jest trudniejsze — należy je traktować jako jeden z dalekosiężnych planów programometrii. Polega ono na zdefiniowaniu reguł obliczania jednej miary związanej z ocenianym systemem. Reguły obliczania tej miary byłyby określone na wszystkich podstawowych czynnikach złożoności wszystkich obiektów programowych tworzących dany system.

W strukturze pojęć programometrii jest miejsce na ocenę takich osiągnięć technologii programowania, jak programowanie strukturalne, styl programowania czy specyfikacje formalne.

W informatyce znane są działania, które można nazwać optymalizacją miar czynników złożoności. Formatowanie tekstu programu, często będące składową procesu translacji, polega na optymalizacji pewnej miary formy modułu i bloku podstawowego. Halstead [7] podaje zalecenia w zakresie zmniejszania rozmiarów bloku podstawowego — jest to optymalizacja miary rozmiaru. Woodward [16] zaleca takie ułożenie bloków podstawowych, żeby liczba węzłów była jak najmniejsza. Jest to zabieg optymalizujący struktury na poziomie modułu.

Programometrię można stosować dynamicznie, badając obiekty utworzone w kolejnych fazach powstawania oprogramowania. Kolejne fazy cyklu istnienia oprogramowania są wyznaczane przez transformacje od specyfikacji zewnętrznej (systemu lub modułu) do gotowych bloków podstawowych.

W przyszłości będzie zapewne możliwe udzielenie odpowiedzi na wiele pytań z dziedziny programometrii, np. czy warto specyfikować role bloków podstawowych? Będzie to, w pewnym sensie, uogólnienie problemu profesjonalizacji programowania, o którym pisał Weinberg [15]. Chodziło mu o odpowiedź na pytanie, czy warto w sposób wyszukany obsługiwać tablice danych, czy też należy szybko napisać program, implementujący prosty algorytm, choć działający wolniej, ale oszczędzający czas pracy programisty. Odpowiedź na pytanie o sens specyfikacji zewnętrznych bloków podstawowych jest to — w kontekście tego

artykułu — odpowiedź na pytanie, czy złożoność specyfikacji bloku podstawowego będzie mniejsza od jego złożoności wewnętrznej.

Programometria, jak starano się wykazać, jest dziedziną: ogólną, właściwie opisującą stan rzeczywistości informatycznej (świadczy o tym możliwość jej stosowania do oceny dotychczasowych osiągnięć technologii programowania), pozwalającą osiągać nowe, ciekawe rezultaty badawcze, a więc — poszerzająca stan dotychczasowej wiedzy o programowaniu.

Bardzo dziękuję docentowi Janowi Borowcowi z IPI PAN za pomoc w trakcie opracowywania tego artykułu — od momentu zasugerowania tematu, przez proces gromadzenia materiałów, aż do lektury wstępnych wersji tekstu.

LITERATURA

[1] Aho A. V., Ullmann J. D.: Principles of Compiler Design. Addison-Wesley, Reading, Massachussets, 1977
[2] Brauer W. (RED).: Tagungsband GI-11. Jahrestagung 1981
[3] De Young G. E., Kampen G. R.: Program Factors as Predictors of Program Readablity Proc. of COMPSAC, 1979, IEEE, str. 668—683
[4] Gasik S.: Programometria (w przygotowaniu)
[5] Gilb T.: Software Metrics. Winthrop Publishers, Cambridge 1977
[6] Gordon R. D.: Measuring Improvements in Program Clarity IEEE Transactions of Software Engineering. Vol. SE-5, 1979, str. 79—90
[7] Halstead M. H.: Elements of Software Science. Elsevier North-Holland, New York, 1977
[8] Hocker H., Itzfeld W. D., Schmidt M., Timm M.: Comparative Descriptions of Software Quality Measures. Gesellschaft für Mathematik und Datenverarbeitung, GmbH, Bonn, Bericht 81, Mar 1984
[9] Joergensen A. H.: Methodology for Measuring the Readability and Modifiability of Computer Programs, BIT, Vol. 20, 1980, str. 394—405
[10] Keutgen H.: Eine Mertik zur bewertung der modularisierung W [2]
[11] Mc Cabe T. J.: A Complexity Measure, IEEE Transactions on Software Engineering, Vol. SE-2, No 4, Dec 1976, str. 308—320
[12] Oviedo E. I.: Control Flow, Data Flow and Program Complexity proc. of the 4TH COMPSAC Conference, IEEE, October 1980, str. 146—152
[13] Van Tassel D.: Praktyka programowania. WNT, Warszawa 1980
[14] Warnier J. D.: Logical Construction of Programs. Van Nostrand Reinhold, New York 1974
[15] Weinberg G. M.: The Psychology of Computer Programming. Van Nostrand Reinhold, New York, 1971
[16] Woodward M. R., Hennell M. A., Hedley D.: A Measure of Control flow Compilexity in Program Text. IEEE Transactions on Software Engineering, Vol. SE-5, No 1, January 1979, str. 45—50.

ZDZISŁAW PŁOSKI
Instytut Informatyki
Uniwersytet Wrocławski

RATFOR — czyli strukturalny FORTRAN (2)

W bieżącym numerze dokończono omawianie instrukcji RATFORU, przedstawiono opinie o języku oraz krótko opisano zasady jego implementacji.

Instrukcje BREAK i NEXT

Instrukcja BREAK wymusza wyjście z pętli. Dotyczy to pętli każdego rodzaju, także pętli FOR. Analogicznie stosuje się instrukcję NEXT, która wymusza kolejną iterację z pominięciem ewentualnych instrukcji występujących po NEXT.

Zarówno instrukcja BREAK jak i NEXT mogą mieć parametr będący numerem poziomu, na którym iteracja ma zostać zakończona (powtórzona), co ma zastosowanie przy zagnieżdżaniu pętli. W praktyce, instrukcji BREAK lub NEXT z parametrem większym od 1 (tj. dotyczącym bieżącego, dynamicznie najbardziej wewnętrznego poziomu) używa się rzadko. Nieobecność parametru powoduje przyjęcie jedności jako jego wartości domyślnej.

Instrukcja FOR

Instrukcja FOR w RATFORZE zasługuje na szczególną uwagę. Rozważmy łatwo czytelny przykład:

```
FOR (J = 1; J < I; J = J + 1) [ # THEN REVERSE
    K = STR(I)
    STR(I) = STR(J)
    STR(J) = K
    I = I — 1
    ]
```

Zgrupowanie obok siebie (tekstowo) wszystkich elementów organizujących powtórzenia znalazło uznanie wśród krytyków RATFORU [5]. Wyrażenia w nawiasach okrągłych powodują odpowiednio: nadanie wartości początkowej zmiennej sterującej, zbadanie warunku zakończenia pętli, aktualizację zmiennej sterującej. Powyższe wyjaśnienia odnoszą się do podanego przykładu. W ogólnym przypadku interpretacja wyrażeń sterujących ratforową pętlą FOR jest obszerniejsza. Semantycznym równoważnikiem struktury

```
FOR (w1; w2; w3) w4
```

jest ciąg następujący:

```
w1
WHILE (w2)
    w4
    w3
    }
```

Nie nakłada się zatem żadnych szczególnych ograniczeń na składnię wyrażeń sterujących, poza warunkiem w2, który powinien być wyrażeniem logicznym. Mogą to być wyrażenia, jak w podanym przykładzie, naśladujące iterację FOR w klasycznym jej rozumieniu, lecz także — wyrażenia nie związane ze sterowaniem żadnej zmiennej w pętli. Wyjście z pętli może być wtedy wykonane przez instrukcję BREAK. Wyrażeniem sterującym [1]) mogą być wreszcie ciągi instrukcji zamknięte w nawiasy [ ], którym struktura FOR zapewni automatyczne powtarzanie. Każde z wyrażeń w1, w2, w3, w4 może być pominięte, z odpowiednimi skutkami: brak inicjowania, brak kontroli warunku zamknięcia pętli, brak aktualizacji, brak „treści" [2]) pętli. W wypadku pomijania wyrażeń zachowuje się średniki separujące. Średnikiem zaznacza się również brak członu w4 w pętli.

Poniżej podano przykłady różnego użycia w RATFORZE pętli FOR.

```
FOR (; I > 0; I = I — 1)
    CALL OUTCH(CHARS(I))
```

Zmienna I zostanie użyta w pętli z wartością, jaką miała w momencie przekazania sterowania do pętli.

```
FOR (N = 0: ; N = B*N + C = DIG0) [
    C = NGETCH(LEXSTR(1), FD)
    IF (C > = LETA&C < = LETZ)
        C = C — LETA + DIG9 + 1
    ELSE IF (C > = BIGA& C < = BIGZ)
```

[1]) Dopuszcza się w tym miejscu rozszerzenie znaczenia terminu „wyrażenie" także na instrukcję

[2]) Cudzysłów jest celowy — wszak treść może być zawarta w którymś z wyrażeń w1, w2, w3

```
    C = C — BIGA + DIG9 + 1
  IF (C < DIG0 ↑ C > = DIG0 + B)
    BREAK
  ]
```

Mimo braku warunku zakończenia pętli wyjście z pętli następuje przez wykonanie instrukcji BREAK.

```
FOR (LEX = GNBTOK(LEXSTR, MAXTOK); LEX = = NEWLINE;
  LEX = GNBTOK(LEXSTR, MAXTOK))
```

Całość działań zawarto w sterującej części pętli.

Pętla DO

Konstrukcja DO, choć banalna, uwalnia użytkownika od obowiązku etykietowania i jawnego zaznaczania końca pętli (przez CONTINUE), jak w FORTRANIE, np.

```
DO I = 1,128
  TABPTR(I) = 0
```

Pętle dowolnego rodzaju mogą być zanurzone jedna w drugiej, a jedynym ograniczeniem jest pojemność stosów organizowanych przez preprocesor.

Instrukcja RETURN

Instrukcja RETURN w RATFORZE niewiele się różni od jej fortranowskiego pierwowzoru. Uogólnienie polega na dopuszczeniu parametru (wyrażenie w nawiasach), który preprocesor przypisuje nazwie funkcji.

```
# EQUAL — COMPARE STR1 TO STR2; RETURN YES IF EQUAL,
  NO IF NOT INTEGER FUNCTION EQUAL(STR1, STR2)
  CHARACTER STR1(ARB), STR2(ARB)
  INTEGER I
  FOR (I = 1; STR1(I) = = STR2(I); I = I + 1)
    IF (STR1(I) = = EOS) [
      EQUAL = YES
      RETURN
      ]
EQUAL = NO
RETURN
END
```

W powyższym opisie instrukcje wyjścia i nadania wartości nazwie funkcji można połączyć pisząc odpowiednio:

```
RETURN(YES)
RETURN(NO)
```

Przeszłość i teraźniejszość RATFORU

Wkrótce po opublikowaniu przez Briana W. Kernighana podstawowego artykułu o RATFORZE recenzenci pisali:

„Choć FORTRAN nie jest najmilszy w użyciu, ma przewagę w uniwersalności i (zwykle) względnej efektywności. Język RATFOR usiłuje ukryć główne niedostatki FORTRANU zachowując jego pożądane zalety, przez zastosowanie przyzwoitych struktur sterowania i trochę „syntaktycznego cukru" (...). Po ukryciu fortranowskich struktur sterowania i jego kosmetycznych niedostatków otrzymujemy język nader miły w użyciu. Programy w RATFORZE są istotnie łatwiejsze do pisania i czytania, a przez to łatwiejsze do uruchamiania, pielęgnowania i modyfikowania niż ich odpowiedniki fortranowskie.

RATFOR umożliwa pisanie programów dających się przenosić na inne instalacje". [1]

„Ten artykuł przywodzi na myśl przysłowie (...): „Lepiej zapalić jedną świecę niż brnąć w ciemnościach" (...) Przy całym uroku RATFORU nie można go wszelako rozpatrywać jako niekwestionowanego sukcesu. W paragrafie zatytułowanym „Konkluzje" autor pisze: "RATFOR ukazuje, że przy umiarkowanym wysiłku można przekształcić FORTRAN ze złego języka w zupełnie niezły" (...). Paragraf ten poprzedza inny, zatytułowany „Rzeczy złe", w którym autor podaje z rozbrajającą szczerością wykaz tych problemów, które spadają na użytkownika RATFORU (...) Sądzę, że niektórzy dostawcy FORTRANU mogą się dozbroić z pożytkiem w RATFOR, przynajmniej do czasu, gdy nowe kompilatory (wg standardu 77 — przyp. ZPL) ujrzą światło dzienne". [8]

W dwa lata po opublikowaniu opisu RATFORU Kernighan „dodał mu skrzydeł" opracowując pakiet oprogramowania nienumerycznego [3]), pod nazwą SOFTWARE TOOLS (narzędzia programowe) i wydając książkę pod tym samym tytułem [5]. Ówczesny recenzent [7] tej książki pozytywnie wyraził się o pakiecie: „Ostateczne wersje programów są bardzo przejrzyste i czytelne". Zważywszy, że ten sam recenzent pisał w innym miejscu: „Pomimo tego, że RATFOR wygląda niezgrabnie dla programisty biegłego w PASCALU lub PL/I, użycie RATFORU w realny sposób pomaga w konstruowaniu i eksponowaniu przejrzystych programów", można nabrać przekonania o przydatności RATFORU.

Narzędzia SOFTWARE TOOLS nobilitowały RATFOR. Z ich pomocą dotrwał on do bieżącego dziesięciolecia. W 1982 r. Department of Computer Science uniwersytetu w Tucson (USA) anonsował kolejną implementację SOFTWARE TOOLS w RATFORZE.

Preprocesor RATFORU tłumaczy na bardzo elementarny podzbiór FORTRANU, co przy rozpowszechnieniu FORTRANU (I miejsce wg danych z przełomu lat 1979—1980) prowadzi do prostego wniosku: programowanie w RATFORZE daje produkty przenośne. Wniosek ten potwierdza praktyka. RATFOR-0, tj. kod RATFORU w FORTRANIE dał się przetłumaczyć bez najmniejszej zmiany na maszynie ODRA 1305 za pomocą kompilatora FORTRANU 1900 (por. uwagi w następnym punkcie).

Przykład implementacji samego RATFORU (także na mikrokomputerach, np. OSBORNE) w RATFORZE, jak również licznych wcieleń pakietu SOFTWARE TOOLS w RATFORZE [4]) ilustruje korzystanie z naturalnej przenośności (przez FORTRAN), jaką daje oprogramowanie napisane w RATFORZE.

Wciąż ekskluzywny, ale zdobywający coraz większą popularność język ICON [2], [3] w swych wczesnych wersjach osiągnął status mobilności właśnie przez RATFOR [5]).

Implementacja RATFORU

Zainstalowanie RATFORU na nowym typie komputera jest przedsięwzięciem stosunkowo łatwym, a w każdym razie dobrze określonym. Zadanie sprowadza się do następujących kroków:

- sporządzenie tablic konwersji kodów znaków dla wejścia-wyjścia (wewnętrznie RATFOR działa w kodzie ASCII)
- przetłumaczenie i przetestowanie dostarczanej w FORTRANIE początkowej wersji RATFORU, za pomocą dostępnego kompilatora FORTRANU (tzw. wersja 0)
- uzyskanie sprawniejszej wersji na podstawie wersji 0 (niezbędnych zabiegów adaptacyjno-optymalizacyjnych można dokonywać już w RATFORZE)
- powtarzanie procesu ulepszania stosownie do potrzeb.

Do uzyskania praktycznych efektów istotne są usprawnienia w pakiecie wejścia-wyjścia RATFORU. Zabiegi adaptacyjne, z uwagi na indywidualne cechy komputera i systemu operacyjnego, są tu na ogół niezbędne. Standardowy pakiet we-wy napisany w FORTRANIE można zastąpić własnym, posługując się kodem wewnętrznym danego komputera w celu zaprogramowania najintensywniej używanych podprogramów. Uzyskiwana w ten sposób oszczędność czasu może przekraczać 50%.

Autor artykułu ma pozytywne doświadczenia z instalowania wersji RATFORU i SOFTWARE TOOLS rozpowszechnianych przez uniwersytet w Tucson. System operacyjny GEORGE 3 pozwala rozwiązać problemy operowania plikami w RATFORZE, a aparat makrogeneracji GEORGE 3 umożliwia sprawne połączenie fazy przetwarzania

---

[4]) Pakiet ten zawiera procedury z zakresu przetwarzania tekstów i manipulowania zbiorami

[4]) Druga książka Kernighana dotycząca SOFTWARE TOOLS, a poświęcona ich wyrażeniu w PASCALU [4], spotkała się z dość ostrą krytyką za używanie w niej PASCALA „po ratforowemu" [6]! W podstawowej publikacji o języku C zrodzone po raz pierwszy w Ratforze SOFTWARE TOOLS znowu wróciły jako materiał ilustracyjny

[5]) Nowe wersje ICONU (obecnie 5.9) rozpowszechnia się w języku C

dokończenie na str. 51

EDWARD BIELENINIK, ELŻBIETA HUDYMA,
ELŻBIETA KOSMULSKA-BOCHENEK,
JÓZEF LEWOC, ANDRZEJ STANISZ
Centrum Obliczeniowe
Politechnika Wrocławska

Podsieć komunikacyjna Międzyuczelnianej Sieci Komputerowej

Podsieć komunikacyjna stanowi podstawowy element składowy sieci komputerowej. Prace nad jej budową zostały zainicjowane w połowie 1979 r., po uruchomieniu tematu badawczego dotyczącego powstania Międzyuczelnianej Sieci Komputerowej (MSK). Przed wykonawcami postawiono wówczas następujące główne zadania podsieci:

- **budowa i wdrożenie do eksploatacji 3-węzłowej podsieci komunikacyjnej**
- **przygotowanie specjalistów projektowania i budowy sieci transmisji danych**
- **opracowanie narzędzi i metod badawczo-pomiarowych sieci komputerowych.**

Prace nad oprogramowaniem węzłów podsieci prowadzono równolegle w Politechnice Wrocławskiej i Politechnice Śląskiej, dokonując niezbędnych uzgodnień, aby zapewnić współpracę obu wersji węzłów. Przyjęte wówczas założenia są spójne z założeniami określonymi dla MSK [2]. Założenia te można sprowadzić do następujących zagadnień:

- wykorzystanie sprzętu krajowego lub krajów RWPG
- oparcie się na międzynarodowych standardach i normach
- budowa od podstaw całego niezbędnego oprogramowania
- zapewnienie łatwej rozbudowy podsieci.

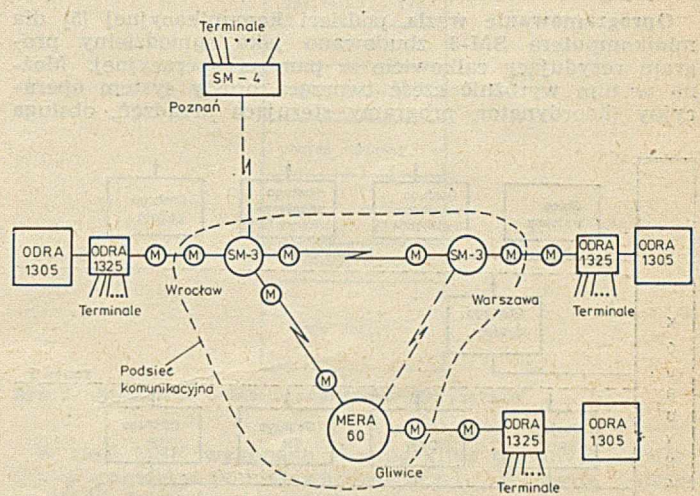
Przyjęto także, że liczba linii, jaką węzeł powinien obsłużyć, nie może być mniejsza niż 6 i że w pierwszym etapie będą to linie o szybkości 2400 b/s.

Główną funkcją węzła jest odbiór i nadawanie informacji od (do) współpracujących abonentów i innych węzłów podsieci. Realizacja tej funkcji jest zazwyczaj połączona z koniecznością spełniania wielu czynności dodatkowych, jak:

- buforowanie danych
- wybór drogi w podsieci
- przeprowadzanie niezbędnych akcji korekcyjnych
- sterowanie i pomiary
- współpraca z operatorem węzła.

STRUKTURA PODSIECI

Konfigurację podsieci komunikacyjnej przedstawiono na rys. 1. Liniami przerywanymi zaznaczono połączenia (nie eksploatowane na bieżąco). Koncentrator terminali znajduje się w fazie testowania. Węzły podsieci są połączone



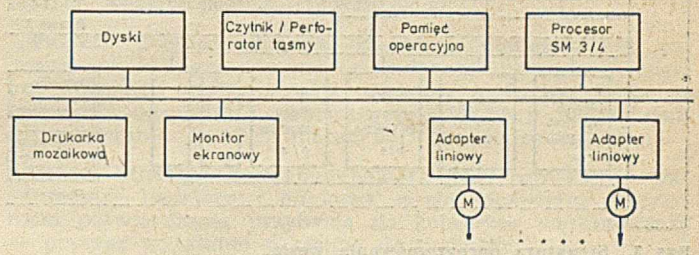
Rys. 1. Konfiguracja podsieci komunikacyjnej

liniami telefonicznymi. Stosowana szybkość transmisji wynosi 2400 b/s.

Terminale są przyłączone do podsieci za pomocą procesorów czołowych (ang. front-end processors) lub poprzez koncentrator. Komputery obliczeniowe ODRA 1305 są włączone do sieci także poprzez procesory czołowe. Mogą one być równocześnie dostępne z lokalnych terminali.

Sprzętową konfigurację węzła tworzą następujące moduły minikomputera SM-3 (rys. 2):

- procesor SM 2401



Rys. 2. Struktura sprzętowa węzła

Mgr inż. EDWARD BIELENINIK ukończył w 1970 r. studia na Wydziale Elektroniki Politechniki Wrocławskiej oraz w 1982 r. na Wydziale Matematyki Uniwersytetu Wrocławskiego. Pracował w Instytucie Telekomunikacji i Akustyki Politechniki Wrocławskiej i ZETO Wrocław. W 1977 r. otrzymał nagrodę Ministra Nauki, Szkolnictwa Wyższego i Techniki. Od 1980 r. pracuje w Centrum Obliczeniowym Politechniki Wrocławskiej, gdzie zajmuje się problemami sieci komputerowych, w szczególności podsieci transmisji danych.

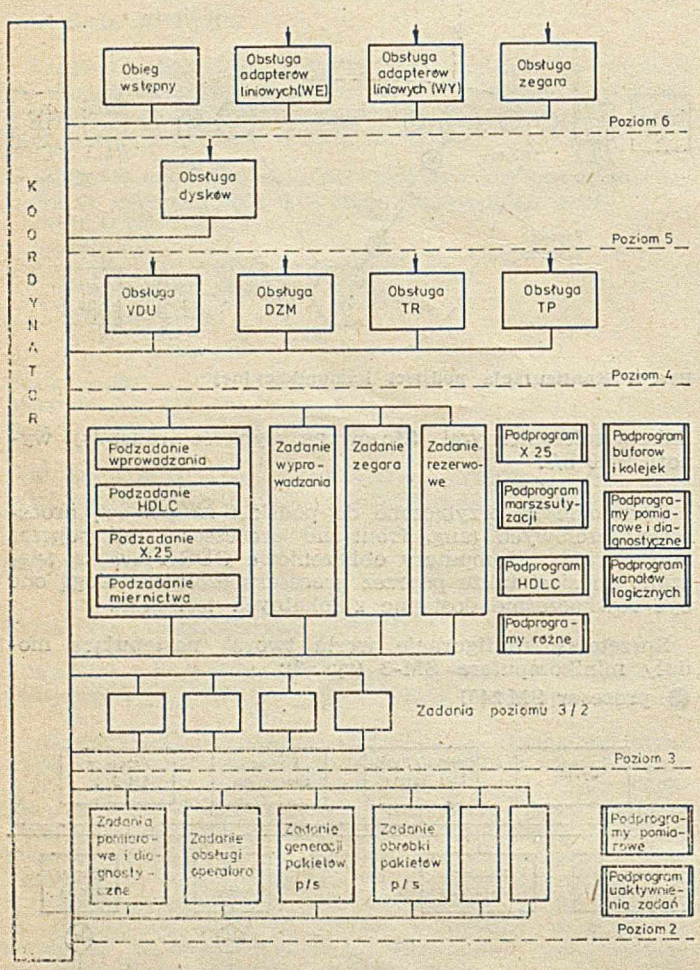
Dr inż. ELŻBIETA HUDYMA ukończyła Wydział Elektroniki Politechniki Wrocławskiej w 1967 r. (kierunek — maszyny matematyczne), a w 1976 r. uzyskała tytuł doktora nauk technicznych. Obecnie pracuje w Centrum Obliczeniowym Politechniki Wrocławskiej jako adiunkt. Zajmuje się problemami diagnostyki systemów cyfrowych.

- pamięć operacyjna 28 k słów
- pamięć dyskowa (2,5 Mb)
- monitor ekranowy
- czytnik taśmy dziurkowanej (perforator taśmy opcjonalnie)
- drukarka mozaikowa (opcjonalnie)
- adaptery liniowe ALS-11.

Adaptery liniowe ALS-11 są kompatybilne z adapterami DUP-11 firmy DEC. Zostały one wykonane w Politechnice Wrocławskiej. Istotnym ograniczeniem tych adapterów jest brak opcji bezpośredniego dostępu do pamięci (DMA), co wymaga obsługi przerwań na poziomie znakowym.

Dyski są używane do celów pomocniczych — organizacji restartu węzła oraz zbierania przez system pomiarowo-diagnostyczny podsieci — danych dotyczących pracy węzła i podsieci.

Oprogramowanie węzła podsieci komunikacyjnej [5] dla minikomputera SM-3 zbudowano jako samodzielny program rezydujący całkowicie w pamięci operacyjnej. Można w nim wyróżnić część tworzącą prosty system operacyjny (koordynator, programy sterujące urządzeń, obsługa operatora) oraz część użytkową (odbiór, przetwarzanie i nadawanie danych, generowanie i przetwarzanie informacji pomiarowo-sterujących itp.).



Rys. 3. Struktura oprogramowania węzła

Funkcje węzła są realizowane w modułach pracujących na różnych poziomach priorytetu (rys. 3), zgodnie z architekturą procesorów rodziny SM, przy czym działania uwarunkowane czasowo (wymagające krótkich czasów reakcji) zlokalizowano na wyższych poziomach priorytetu (np. obsługa przerwań adapterów liniowych i urządzeń zewnętrznych), natomiast pozostałe — o słabszych uwarunkowaniach czasowych — na niższych. Poziomy priorytetu 6, 5 i 4 przeznaczono do obsługi przerwań zewnętrznych, poziomy 3 i 2 — do przetwarzania o różnym stopniu pilności.

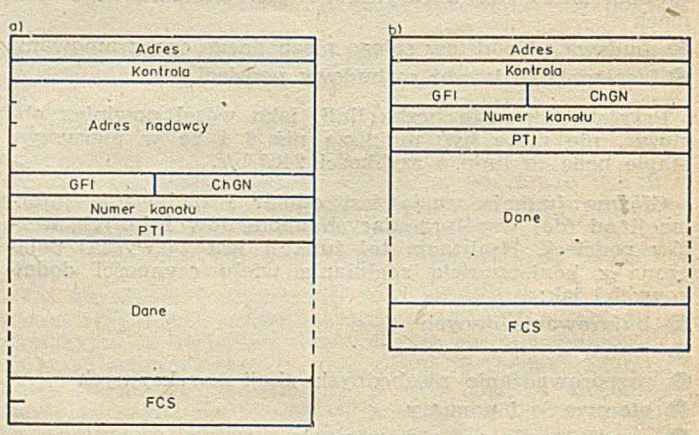
W strukturze programu wyodrębniono zadania kilku typów, będące zamkniętymi częściami programu i spełniającymi określoną funkcję. Zadania są zwykle uruchamiane przez koordynator, niekiedy — przerwaniem zewnętrznym, a po zakończeniu pracy oddają sterowanie do koordynatora. Zadania współpracują ze sobą poprzez koordynator, mogą się wzajemnie uaktywniać, przekazywać dane oraz korzystać z zestawu podprogramów.

Program węzła zajmuje ok. 22 k słów pamięci operacyjnej, pozostawiając ok. 6 k słów na struktury danych (opisy linii, opisy połączeń logicznych) oraz bufory.

Biorąc pod uwagę potrzebę rozszerzenia funkcji wybranego węzła o zadania pomiarowo-sterujące podsieci, wykonano wersję nakładkową systemu operacyjnego węzła. W realizacji tej zadania pomocnicze wykonywane na poziomie drugim będą rezydować na dysku, co pozwoli na znaczną rozbudowę oprogramowania węzła, szczególnie o funkcje pomiarowo-sterujące. Docelowo dla węzłów pomiarowo-sterujących oraz o dużym ruchu będzie stosowany SM-4.

REALIZOWANE USŁUGI

Protokół zewnętrzny i wewnętrzny zastosowany w podsieci komunikacyjnej jest oparty na zaleceniach CCITT [3,6] — X.21 bis na poziomie fizycznym i X.25 na poziomach liniowym i pakietowym.



Rys. 4. Format ramek informacyjnych
a) ramka międzywęzłowa; b) ramka na styku do DTE
GFI — General Format Identifier (identyfikator formatu)
ChGN — Channel Group Number (numer grupy kanałów)
PTI — Packet Type Identifier (identyfikator typu pakietów)
FCS — Frame Check Sequence (kod kontrolny)

Dr inż. ELŻBIETA KOSMULSKA-BOCHENEK ukończyła Wydział Elektroniki Politechniki Wrocławskiej. Tytuł doktora nauk technicznych uzyskała w 1972 r. Pracowała w Katedrze Konstrukcji Maszyn Cyfrowych, następnie w Instytucie Cybernetyki Technicznej, a obecnie w Centrum Obliczeniowym Politechniki Wrocławskiej na stanowisku adiunkta. Zajmuje się problemami diagnostyki i niezawodności systemów cyfrowych.

Mgr inż. ANDRZEJ STANISZ ukończył studia na Wydziale Elektroniki Politechniki Wrocławskiej. W latach 1970—1979 pracował w Instytucie Cybernetyki Technicznej, a od roku 1979 — w Centrum Obliczeniowym Politechniki Wrocławskiej. Zajmuje się zagadnieniami sieci komputerowych i oprogramowaniem minikomputerów z rodziny SM.

Jako protokół międzywęzłowy wykorzystano zmodyfikowaną wersję protokołu X.25 (rys. 4). W nagłówku ramki przesłany jest dodatkowo adres DTE (Data Terminal Equipment) nadawcy (8 bajtów). Ułatwia to działanie odpowiednich procedur w węźle, ale zabiera 8 znaków z pasma przepustowości linii.

Dodatkowe elementy w ruchu międzywęzłowym stanowią pakiety pomiarowo-diagnostyczne. Są one identyfikowane w nagłówku pakietu za pomocą pola GFI (General Format Identifier), (które wynosi 1111) i rozróżniane między sobą w oparciu o rozszerzenia pola GFI. Pakiety pomiarowo-diagnostyczne są zatem przesyłane na zasadzie datagramów. Celem uniknięcia zapętlenia, dopuszczalna liczba przejść międzywęzłowych jest ograniczona.

Usługi podstawowe podsieci na styku do DTE wynikają z protokółu X.25. W aktualnej wersji podsieci wdrożono następujące opcje:

- połączenia wirtualne (stałe i czasowe)
- połączenia typu końcowego (ang. end-to-end)
- standardowa długość pakietów danych wynosi 128 bajtów
- numeracja pakietów jest liczona modulo 8
- negocjacja rozmiaru okna.

Prowadzone są działania zmierzające do rozdziału dwóch trybów pracy: interakcyjnej i przesyłania zbiorów. Powinno to w widoczny sposób poprawić sprawność podsieci [4]. Dla ułatwienia obsługi podsieci opracowano wewnętrzny system pomiarowo-diagnostyczny pozwalający na rejestrację ważniejszych zdarzeń, a także wprowadzono zestaw pakietów pomiarowo-diagnostycznych jako usługi dodatkowe.

System pomiarowo-sterujący (SPS) ma strukturę modularną, która pozwala na wygenerowanie pewnego podzbioru modułów tego systemu. Funkcje spełniane przez SPS podzielić można na lokalne i rozproszone.

System lokalny zamyka się całkowicie w ramach węzła. Głównym jego zadaniem jest zbieranie danych o pracy węzła, składowanie ich na dysku oraz monitorowanie. Przetwarzanie pamiętanych na dysku danych pomiarowych wykonywane jest w trybie pośrednim. W systemie lokalnym nie ma mechanizmów wymiany informacji między węzłami, natomiast przetwarzanie danych pomiarowych daje statystyczny obraz działania węzła. Zebrane i zapamiętane w systemie lokalnym dane o pracy węzła tworzą podstawową bazę informacyjną, wykorzystywaną dalej w systemie rozproszonym. Baza ta powstaje w oparciu o dwa mechanizmy: rejestrację zdarzeń i rejestrację stanów.

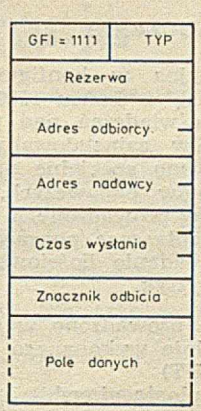
Rejestracja zdarzeń polega na odnotowaniu faktu wystąpienia określonego zdarzenia w pracy węzła. Przykładami zdarzeń istotnych dla pracy węzła są: nawiązanie połączenia, wysłanie określonego typu pakietu, przyjęcie ramki, wyłączenie linii, restart węzła itp. Dobór zdarzeń do rejestracji uwarunkowany jest potrzebami wynikającymi zarówno z eksploatacji MSK, jak i z prac badawczych prowadzonych nad podsiecią.

Najprostszym sposobem rejestracji jest zwiększenie o jedność licznika przyporządkowanego danemu zdarzeniu w momencie wystąpienia tego zdarzenia. W pewnych wypadkach, oprócz zliczania, rejestruje się dodatkowe atrybuty zdarzeń, co przy natychmiastowym przetwarzaniu daje opis zdarzenia w postaci danych statystycznych, takich jak wartość średnia czy maksymalna parametru. Liczbę przetwarzanych parametrów oraz liczbę danych statystycznych ogranicza czas przetwarzania, który musi być tu bardzo krótki.

Rejestracja zdarzeń polega na okresowym notowaniu wartości chwilowej określonych parametrów, charakteryzujących stan węzła. Wartości te przechowywane są w pamięci do chwili następnego pomiaru. Niektóre parametry podlegają bieżącemu przetwarzaniu, dzięki któremu w każdej chwili dostępna jest ich wartość średnia i maksymalna. Moment rejestracji zdarzeń wyznacza zegar lub żądanie operatora. Częstość rejestracji zależy od szybkości zmian wartości parametru.

System rozproszony pozwala na wymianę informacji pomiarowo-sterującej pomiędzy węzłami. Mechanizmem wymiany są pakiety pomiarowo-sterujące. W każdy węzeł wbudowane są moduły umożliwiające tworzenie, nadawanie i odbiór pakietów. Pakiety inicjowane są przez operatora dowolnego węzła żądającego informacji z podsieci. W tym celu zdefiniowany został prosty język poleceń, pozwalający na utworzenie i wysyłanie w podsieć pakietów. Odpowiedzi na wysyłane pakiety monitorowane są w węźle generującym pakiet.

Struktura pakietów pomiarowo-sterujących została zaprojektowana zgodnie z pakietami protokołu X.25 [1], obowiązującego w sieci MSK (rys. 5). Różnią się one kodem GFI, który ma postać 1111 dla pakietów systemu SPS. Nagłówek pakietu poza kodem GFI oraz typem pakietu zajmującym drugą połowę oktetu zawiera adres odbiorcy, adres nadawcy, czas wysyłania pakietu oraz znacznik odbicia. Znacznik ten przyjmuje wartość 1 w pakietach nadawanych (pierwotnych), a wartość 0 w pakietach odpowiedzi. Poniżej nagłówka jest pole danych, którego rozmiar i znaczenie zależy od typu pakietu. Długość pakietu pomiarowego nie może być większa niż maksymalna długość standardowego pakietu w MSK.



Rys. 6. Struktura pakietu pomiarowo-sterującego

W sieci MSK wykonano obecnie kilka opisanych poniżej typów pakietów pomiarowo-sterujących. W miarę potrzeby liczba ta może ulec zwiększeniu.

Pakiet ECHO. Zadaniem pakietu jest sprawdzenie możliwości uzyskania połączenia z wybranym węzłem sieci zgodnie z marszrutyzacją. Dodatkową funkcją pakietu jest rejestracja czasu przybycia pakietu do odbiorcy, co daje możliwość określenia charakterystyk czasów opóźnień na poszczególnych trasach.

Pakiet śledzenia drogi (TRACE). Pakiet ten daje obraz drogi, jaką przebył pakiet od węzła nadawcy do węzła docelowego. Kolejne węzły pośredniczące wpisują do pakietu swoje adresy zarówno na trasie pierwotnej, jak i powrotnej.

Pakiet drogi wymuszonej (ROUTE). Nadawca wymusza trasę pakietu do określonego węzła, poprzez podanie kolejnych węzłów pośredniczących drogi.

Pakiety zbierania danych. Zadaniem pakietów jest pobranie określonych danych z zadanego obszaru pamięci adresowanego węzła. Typów pakietów może być kilka w zależności od rodzaju pobieranych informacji lub miejsca ich pobierania (pamięć operacyjna, pamięć zewnętrzna).

Pakiet SONDA. Zadaniem pakietu jest sprawdzenie trasy między nadawcą i odbiorcą, z równoczesnym wysyłaniem potwierdzenia przybycia do kolejnych węzłów trasy — poprzez wysyłanie pakietów do nadawcy.

OBSŁUGA OPERATORSKA

Obsługa operatorska węzła powinna być ograniczona do minimum. Jednak na etapie wstępnej eksploatacji sieci jest ona niezbędna. Do podstawowych czynności operatora należy załadowanie węzła, obsługa stanów awaryjnych i zakończenie pracy. Do dyspozycji operatora przewidziano kilkanaście poleceń ułatwiających współpracę z węzłem. Umożliwiają one wykonanie takich czynności, jak przydział urządzeń zewnętrznych, zmiana daty i czasu, monitorowanie stanu węzła, wysyłanie pakietów pomiarowo-sterujących.

Do podstawowych **parametrów ilościowych węzła** należy przepustowość i opóźnienie tranzytowe. Z dokonanej analizy oraz pomiarów wynika, że czas przetwarzania ramki w węźle jest rzędu kilku milisekund i jest minimalny w

porównaniu z czasem transmisji (zwłaszcza przy szybkości linii 2400 i 9600 b/s). Opóźnienie w węźle może odgrywać istotną rolę w pobliżu progu jego nasycenia (kolejki do nadajnika linii).

Przepustowość węzła jest wyrażana w b/s lub ramkach /s. Z wartości przepustowości wynika liczba linii, jaką węzeł może obsłużyć. Obliczono szacunkową przepustowość węzła dla minikomputerów SM-3 i SM-4, pracujących z adapterami liniowymi ALS-11 i ALS-11M. Adapter ALS-11M ma mechanizm bezpośredniego dostępu do pamięci, natomiast ALS-11 wymaga obsługi przerwań znakowych. Dzięki tym obliczeniom nie są gubione przerwania w niekorzystnych okolicznościach (przesyłanie ciągu krótkich ramek). Wynika z nich, że węzeł może obsłużyć od ośmiu (SM-3/ALS-11) do 20 linii (SM-4/ALS-11M) przy szybkości 2400 b/s.

UZYSKANE DOŚWIADCZENIA

Podsieć komunikacyjna w konfiguracji przedstawionej na rys. 1 zaczęła pracę w końcu 1983 r. Wcześniej przeprowadzono szereg doświadczeń nad pracą w warunkach laboratoryjnych węzłów zainstalowanych w jednym pomieszczeniu. Opracowano specjalne narzędzia pomiarowo-diagnostyczne do celów badawczych. Zasadniczo można wyróżnić dwa rodzaje prowadzonych badań. Pierwszy, to badania jakościowe, dotyczące poprawności zastosowanych rozwiązań, drugi — badanie ilościowe dotyczące pomiaru wybranych charakterystyk.

Badania jakościowe prowadzono w dwóch wariantach:
- za pomocą specjalnie wykonanego Modułu Intensywnego Testowania (MIT)
- w warunkach eksploatacyjnych.

W pierwszym wypadku testowany był w warunkach intensywnego ruchu węzeł uzupełniony o moduły MIT (uproszczone DTE), w drugim — podsieć w warunkach zbliżonych do rzeczywistej eksploatacji.

Także badania ilościowe węzła i podsieci prowadzone były w dwóch wariantach:
- za pomocą Systemu Intensywnego Testowania (SITWA)
- za pomocą Modułu Operacyjno-Pomiarowego (MOP).

System SITWA został opracowany wychodząc z symulacyjnej metody testowania węzła. Z kolei w oparciu o MIT został wykonany MOP. Istotna różnica między obu systemami polega na tym, że SITWA stanowi uzupełnienie oprogramowania węzła o moduły symulujace ruch i moduły pomiarowe, natomiast MOP stanowi rodzaj DET zlokalizowanego poza węzłem.

Przeprowadzone badania i bieżąca eksploatacja podsieci potwierdziły słuszność przyjętych założeń i poprawność zastosowanych rozwiązań. Pozwoliły one na określenie właściwych procedur techniczno-organizacyjnych eksploatacji i opracowanie dokumentacji eksploatacyjnej. Sformułowano także szereg wniosków dotyczących weryfikacji niektórych procedur, szczególnie tych związanych z obsługą, diagnostyką i sterowaniem pracą węzłów i podsieci.

Planowany rozwój podsieci

Przedstawione dalej zamierzenia rozwojowe podsieci nie dotyczą rozbudowy jej konfiguracji (zwiększenie liczby węzłów itp.), lecz wzbogacenia oprogramowania. Przy założeniu, że podsieć komunikacyjna na minikomputerach SM-3 i MERA 60 będzie eksploatowana w ciągu najbliższych kilku lat, oprogramowanie węzłów wymaga uzupełnienia i drobnych korekt. Ich wdrożenie powinno poprawić efektywność pracy podsieci i ułatwić jej eksploatację. Szczególnie niezbędna — z punktu widzenia organizacji eksploatacji MSK — jest rozbudowa procedury pomiarowo-diagnostycznej.

Proponowane prace można podzielić na kilka grup tematycznych. Są to:
- procedury pomiarowo-diagnostyczne i sterujące
- sprawozdawczość dotycząca ruchu węzłów i podsieci
- system automatycznego informowania operatorów
- optymalizacja rozwiązań i niezbędne modyfikacje wynikłe z potrzeb bieżącej eksploatacji.

Najważniejsza, z punktu widzenia eksploatacji sieci, jest rozbudowa systemu pomiarowo-sterującego.

Przyjęto koncepcję hierarchicznej struktury systemu pomiarowo-sterującego, co umożliwia rozbudowę systemu przy wykorzystaniu mechanizmów wprowadzonych na niższym poziomie. Pierwszy poziom to system lokalny, drugi — system rozproszony. Najważniejszym planowanym poziomem będzie system centralny, gdzie występować będzie jednostka nadrzędna systemu pomiarowo-sterującego. Jednostką taką jest centrum pomiarowo-sterujące (CPS), którego funkcję pełnić może jeden wyróżniony węzeł lub niezależny minikomputer przeznaczony wyłącznie do celów diagnostyki i sterowania siecią. Wyróżnienie to może być konsekwencją fizycznego przystosowania jednego węzła do pracy jako CPS, poprzez wyposażenie go w odpowiednie mechanizmy sprzętowe i programowe. Umożliwiają one pracę tego typu lub utrzymywanie potencjalnej gotowości do pracy w charakterze CPS pewnej grupy, a nawet wszystkich węzłów z nadaniem czynnych uprawnień w danej chwili jednemu węzłowi.

Niezależnie od przyjętej metody wyboru węzła centralnego, wszystkie pozostałe muszą mieć zdolność komunikowania się z węzłem nadrzędnym w sensie interpretacji poleceń, ich wykonywania i przekazywania odpowiedzi. Wymiana informacji pomiędzy węzłem nadrzędnym i resztą odbywa się poprzez pakiety kontrolno-pomiarowe. Funkcje spełniane przez węzeł centralny dotyczą możliwości zbierania i przetwarzania danych z całej podsieci oraz podejmowania odpowiednich decyzji zarówno co do sterowania diagnostyką, jak i przekazywania danych do sterowania dla systemu operacyjnego sieci (np. do programu marszrutyzacji). Centrum pomiarowe może dysponować możliwościami zdalnego odczytu (zapisu dowolnego miejsca w pamięci określonego węzła), zdalnego ładowania poprzez podsieć wybranej wersji oprogramowania węzła, aktualizacji czasu w całej podsieci itp.

Sterowanie podsiecią poprzez system hierarchiczny powoduje wzrost efektywności pracy oraz niezawodność sieci. Z drugiej jednak strony, w ocenie systemu nie należy pomijać podstawowej wady scentralizowanego sterowania, jaką jest niska niezawodność samego systemu diagnostyczno-pomiarowego podsieci. Dodatkowa niedogodnością jest możliwość przeciążenia linii w sieci pakietami diagnostyczno-pomiarowymi.

LITERATURA

[1] Bieleninik E., Hudyma E., Kosmalska-Bochenek E., Lewoc J. Stanisz A.: Koncepcja systemu pomiarowo-sterującego podsieci komunikacyjnej Międzyuczelnianej Sieci Komputerowej. Raport CO PWr., Seria SPR 3/83, Wrocław 1983

[2] Bilski E.: Projekt techniczno-funkcjonalny Międzyuczelnianej Sieci Komputerowej — cz. I, ICT i CO PWr., Raport 15/82, Wrocław, 1982

[3] CCITT Recommendation, Yellow Book, Vol. VIII, Fasc. VIII.2, November 1980

[4] Lewoc J., Bieleninik E., Stanisz A.: Influence of Bulle Transfer and Interrogating Mode Traffic Separation on Performance of Computer Networks. Networks in Office Automation, Sophia 1984

[5] Praca zbiorowa: Zagadnienia oprogramowania węzła podsieci komunikacyjnej sieci komputerowej MSK. Prace Naukowe CO PWr., Seria Studia i Materiały 2/1983, Wrocław 1983

[6] Propozycja norm dla podsieci komunikacyjnej Międzyuczelnianej Sieci Komputerowej. Raport CO PWr., Seria SPR, 6/84, Wrocław 1984.

Ogłoszenie • Ogłoszenie • Ogłoszenia • Ogłoszenia

Oprogramowanie do COMMODORE C-64 piszemy na zlecenie instytucji, osób prywatnych. 27-400 Ostrowiec, skrytka 40.

EO/1103/K/85

Pamięci EPROM 2764D oraz statyczne i dynamiczne RAM 16K, 64 K odstąpię. Gdańsk, tel.: 37-43-00.

EO/1433/K/85

Ogłoszenie • Ogłoszenie • Ogłoszenie • Ogłoszenie

KRYSTYNA KOLEŚNIK
JAN SMOLIRA
Centrum Obliczeniowe
Politechnika Wrocławska

Zdarzeniowy język symulacyjny PASCALSIM i jego implementacja

Język symulacyjny PASCALSIM jest rozszerzeniem języka PASCAL o standardowe, charakterystyczne dla symulacji, typy danych i funkcje.

Język ten zdefiniowano w celu udogodnienia symulacji w PASCALU. Prace nad jego implementacją prowadzono od 1981 r. w Centrum Obliczeniowym Politechniki Wrocławskiej w ramach prac dyplomowych i projektu z symulacji. Aktualna wersja języka i translatora jest stosowana praktycznie do programowania problemów symulacyjnych w ramach projektu do przedmiotu „Języki i programowanie — symulacja" na IV roku Wydziału Informatyki i Zarządzania Politechniki Wrocławskiej.

Język PASCALSIM jest językiem ukierunkowanym na programowanie zdarzeń. Jest to wprawdzie jedna ze starszych metod symulacji, najbardziej poglądowa na etapie wstępnego nauczania metod symulacji. Stosowanie języka PASCALSIM pozwala na uproszczenie programowania symulacji w PASCALU i skoncentrowanie uwagi studenta na programowaniu symulowanego problemu, odciążając go od uciążliwego programowania i uruchamiania funkcji pomocniczych.

PASCALSIM nie narzuca istotnych ograniczeń na wykorzystywanie mechanizmów PASCALA i przyzwyczajeń nabytych przy programowaniu w tym języku.

CHARAKTERYSTYKA JĘZYKA

Stworzone przez PASCALSIM udogodnienia dla programowania symulacji umożliwiają łatwy opis statycznej i dynamicznej struktury modelu. W języku rozróżnia się dwa typy obiektów symulacyjnych modelu:

- obiekty trwałe, reprezentujące stałe elementy symulowanego systemu
- obiekty tymczasowe, istniejące w pewnym odcinku czasu działania modelu.

Zdarzenia zachodzące w systemie są związane z przemieszczaniem się obiektów tymczasowych pomiędzy obiektami trwałymi. Atrybuty opisujące obiekty trwałe mogą być opisywane w standardowym PASCALU za pomocą zmiennych i rekordów. Dla obiektów tymczasowych muszą być zdefiniowane typy rekordowe opisujące ich atrybuty, przy czym nazwy tych typów muszą być wyspecyfikowane jako nazwy obiektów tymczasowych przez wymienienie ich po słowie ENTITY na zakończenie definicji typów programu.

W PASCALSIMIE zdefiniowano także standardowe typy służące do organizacji kolejek i wyznaczania charakterystyk modelu. Są to: typ kolejkowy QUEUE oraz typy statystyczne, reprezentujące miary wyznaczone w procesie symulacji, takie jak: zmienne losowe (typ RAND), procesy stacjonarne (typ STAT), procesy niestacjonarne (typ STOCH).

Typ kolejkowy definiuje nagłówek listy i organizuje dostęp do listy złożonej z elementów zawierających odsyłacze do obiektów tymczasowych.

Typy statystyczne reprezentują struktury danych, potrzebne do wyznaczenia uogólnionych parametrów statystycznych dla zmiennych. Dla każdej zmiennej statystycznej wyznaczane są odpowiednie do typu statystycznego zmiennej: wartość średnia, odchylenie standardowe, wartość minimalna i maksymalna oraz opcjonalnie histogram.

Zmienne reprezentujące czas symulacji oraz nagłówek listy kalendarza zdarzeń, wykorzystywane do symulowania upływu czasu, są zmiennymi standardowymi języka.

Zbiór instrukcji języka umożliwa wykonywanie działań odnoszących się do kalendarza zdarzeń oraz działań na zmiennych typów symulacyjnych dostarczanych przez język, reprezentujących obiekty tymczasowe, kolejki i zmienne statystyczne.

Instrukcje kalendarza zdarzeń, złożonego z chronologicznie uporządkowanych opisów zdarzeń, umożliwiają:

- dołączanie do kalendarza opisu zdarzenia określonego typu, związanego ze wskazanym obiektem, zachodzącego w określonym czasie
- zaplanowanie zdarzenia próbkowania dla zmiennych statystycznych typu STOCH
- symulowanie upływu czasu, polegające na pobraniu pierwszego zdarzenia z kalendarza, przesunięciu czasu symulacji do czasu występowania tego zdarzenia oraz na wywołaniu i wykonaniu procedury obsługi zdarzenia.

Przyjęty sposób organizacji upływu czasu wymaga wyróżnienia procedur obsługi zdarzeń przez poprzedzenie ich deklaracji słowem EVENT.

Instrukcje działania na obiektach tymczasowych i kolejkach umożliwiają:

- tworzenie i kasowanie obiektów tymczasowych
- dopisywanie ich na koniec lub początek wskazanej kolejki
- przeszukiwanie kolejki według określonej wartości jednego z atrybutów obiektu.

Instrukcje działania na zmiennych statystycznych umożliwiają:

- aktualizowanie rekordu statystycznego przy zmianach wartości zmiennej statystycznej
- wyprowadzanie wartości uogólnionych parametrów statystycznych i opcjonalnie histogramu zmiennej, po zakończeniu przebiegu symulacji.

Język dostarcza także instrukcji do ustalania stanu modelu, tj. instrukcji zerowania zmiennych statystycznych, pozwalającej na rozpoczęcie zbierania danych w dowolnym momencie pracy modelu oraz instrukcji inicjowania powodującej wyzerowanie czasu symulacji, kolejek, kalendarza zdarzeń i zmiennych statystycznych oraz ustalenie stanu początkowego zmiennych standardowych.

Język jest wyposażony w standardowe funkcje generowania liczb losowych o różnych rozkładach prawdopodobieństwa oraz funkcję podającą długość wskazanej kolejki.

dokończenie na str. 37

Obserwowana na polskim rynku inwazja mikrokomputerów kompatybilnych z IBM PC uzmysłowiła wielu konstruktorom, że już najwyższa pora zająć się konstrukcjami 16-bitowymi. Właśnie kariera komputera osobistego wypuszczonego przez informatycznego kolosa powoduje, że mikroprocesor 8086 ma olbrzymią szansę zdominować wszelkie próby podejmowane przez projektantów. Narzucana konstruktorom zatrudnionym w firmach państwowych konieczność opierania się na podzespołach produkowanych w krajach RWPG sprawiła, że wobec zapowiedzi z ZSRR, pierwsze wprowadzane do produkcji mikrokomputery 16-bitowe opierają się właśnie na 8086. W poniższym tekście nie napisano, dlaczego w krajach przodujących nie stosuje się już 8086 w projektowanych konstrukcjach. Wymagałoby to bowiem szczegółowego porównania z nowszymi mikroprocesorami (np. 68000, 80186, 80286 i in.). Lawina już ruszyła i mikroKLAN jej nie powstrzyma, na nic więc zdałoby się wykazywanie, że istnieją lepsze układy. Skoro jednak 8086 będzie wykorzystywany w krajowych konstrukcjach, warto poświęcić mu trochę miejsca. Poniższe uwagi powinny ułatwić opracowywanie koncepcji przyszłych systemów w taki sposób, by „cienie" były jak najmniej bolesne, a „blaski" posuwały nas choć trochę do przodu.

Blaski i cienie mikroprocesora 8086

Mikroprocesor 8086 jest jednym z pierwszych mikroprocesorów 16-bitowych. W porównaniu z wcześniejszymi mikroprocesorami 8-bitowymi cechuje go znacznie większa moc przetwarzania, stwarzająca możliwość wykorzystywania w zastosowaniach, które przedtem były realizowane w oparciu o minikomputery.

Jednak, jak każde pionierskie rozwiązanie, zawiera on wiele niefortunnych cech, utrudniających jego wykorzystanie. W kolejnych układach mikroprocesorów „spokrewnionych" z 8086 — a więc układach 80186 i 80286 — niektóre z tych błędów starano się wyeliminować.

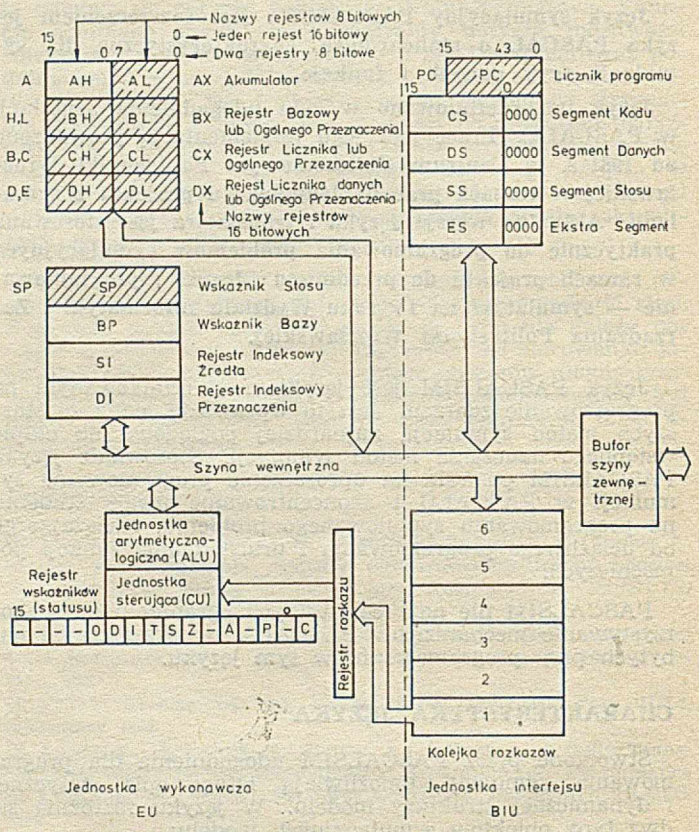
Mikroprocesor 8086 ma identyczną architekturę jak 8088. Różnica polega jedynie na szerokości szyny danych, która dla mikroprocesora 8086 jest 16-bitowa, a dla 8088 — 8-bitowa. Zastosowanie 8-bitowej szyny danych upraszcza budowę systemu, ale kosztem wydłużenia czasu, w jakim realizowane są wszelkie operacje zapisu i odczytu. Dalsze uwagi o mikroprocesorze 8086 można w równym stopniu odnosić do układu 8088.

Dość istotną cechą 8086 jest pewne podobieństwo do popularnego 8080. Szeroko reklamowana jest m. in. kompatybilność języka ASSEMBLER 8086 z językiem ASSEMBLER 8080. Ma to oznaczać, że zestaw rozkazów mikroprocesora 8080 jest podzbiorem zestawu rozkazów 8086. Kompatybilność ograniczona jest jednak do postaci źródłowej programów. Kod maszynowy 8080 nie może być bezpośrednio wykorzystany przez mikroprocesor 8086. Również kompatybilność na poziomie wersji źródłowych programów nie jest pełna. Mnemoniki języka ASSEMBLER 8086 są w przeważającej części nieco inne od stosowanych dla analogicznych operacji w języku ASSEMBLER 8080. Tym niemniej koncepcja logiczna obu języków jest identyczna. W rezultacie zapis symboliczny programu w języku ASSEMBLER 8086 jest podobnie mało przejrzysty jak dla 8080. Jako przykład języka symbolicznego bardziej przejrzystego i prostszego do opanowania pamięciowego można podać ASSEMBLER Z80.

Zestaw rozkazów 8080 nie jest w pełni podzbiorem języka ASSEMBLER 8086. Przykładowo, rozkazy warunkowego powrotu z podprogramu w przypadku 8086 muszą zostać zastąpione przez dwa rozkazy: skoku warunkowego i powrotu.

Programy ASSEMBLER dla 8086 są zazwyczaj tak skonstruowane, że rozpoznają również mnemoniki ASSEMBLERA 8080 i generują kod wynikowy odpowiedniego rozkazu 8086 (lub kilku rozkazów).

Podstawową nowością w strukturze 8086 jest oczywiście możliwość wykonywania operacji na słowach o długości 16 bitów. Dla zapewnienia kompatybilności z 8080 wprowadzono jednak możliwość podziału rejestrów ogólnego zastosowania na 8-bitowe części. Na rysunku 1 pokazano zestaw rejestrów wewnętrznych 8086.



Rys. 1. Architektura mikroprocesora 8086/88. Na rysunku wydzielono linią przerywaną bloki funkcjonalne EU i BIU. Znaczenie poszczególnych bitów w rejestrze wskaźników:
0 — przekroczenie zakresu, D — kierunek transferu, I — przerwania odblokowane/zablokowane, T — pułapka, S — znak, Z — zero, A — przeniesienie pośrednie, P — parzystość, C — przeniesienie

Projektanci 8086 wydzielili dwa bloki funkcjonalne tworząc w strukturze mikroprocesora jak gdyby dwa „podprocesory" (rys. 1). Jeden z nich nazwany został jednostką interfejsu BIU (ang. Bus Interface Unit). BIU jest odpowiedzialny za wszelkie kontakty ze światem zewnętrznym. Drugi to jednostka wykonawcza EU (ang. Execution Unit). Rozpoznaje ona pobrane z pamięci przez BIU rozkazy i steruje ich wykonaniem. Obie części współpracują ze sobą w sposób asynchroniczny na zasadzie przekazywania zleceń i potwierdzeń.

Jednostka BIU wprowadza do 6-bitowej kolejki kody i argumenty kolejnych rozkazów aż do jej zapełnienia. W tym samym czasie jednostka EU zajęta jest zdekodowaniem i realizacją wcześniej wprowadzonego rozkazu. Jeżeli EU stwierdzi konieczność wykonania dodatkowej operacji odczytu lub zapisu, wystawia odpowiednie zlecenie dla BIU, które je realizuje natychmiast po zakończe-

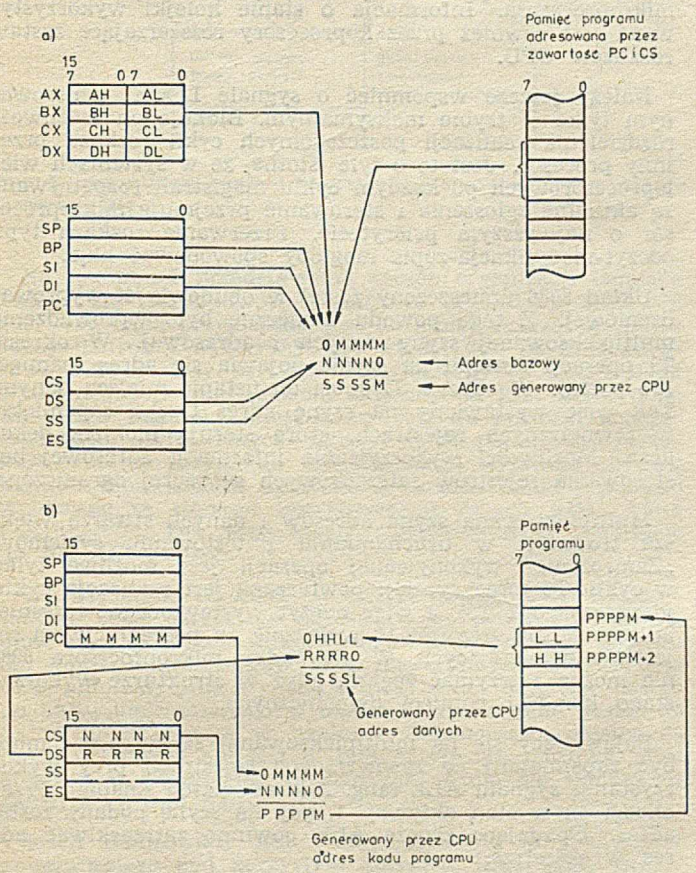
niu aktualnie wykonywanego cyklu. Przyjęcie takiego rozwiązania w znacznym stopniu eliminuje czas martwy, w którym CPU oczekuje na wprowadzenie kodu rozkazu.

Jeżeli w programie wystąpi rozkaz skoku, to kolejka będzie ładowana od początku, a EU musi zaczekać do chwili aż zostanie wprowadzony kod pierwszej operacji z nowego obszaru. Podobnie, odwołanie do informacji, którą trzeba odczytać z pamięci, wstrzymuje pracę EU do momentu aż BIU zrealizuje zlecenie.

BIU przechodzi w stan pasywny tylko wtedy, gdy zapełniona jest kolejka i nie ma żadnych zleceń od EU. Ogranicza to możliwość stosowania tzw. przezroczystych transmisji DMA lub cykli odświeżających pamięci dynamiczne. Polegają one na podkradaniu cykli (ang. cycle stealing), w których CPU zajęte jest dekodowaniem rozkazu i nie realizuje operacji dostępu do pamięci. W ten sposób transfer informacji między urządzeniami wejścia-wyjścia a pamięcią lub cykl odświeżania pamięci dynamicznych nie zakłóca realizacji programu.

W przypadku mikroprocesora 8086 pasywne cykle BIU występują w sposób nieregularny, co praktycznie eliminuje możliwość wykorzystania ich przy transmisji DMA lub do odświeżenia pamięci. Ponieważ jednak i dla wcześniejszych mikroprocesorów firmy INTEL realizacja wspomnianych rozwiązań nie była łatwa, należy uznać, że wprowadzenie kolejki rozkazów jest ulepszeniem zwiększającym efektywność pracy systemu. Typowo, realizacja rozkazu wymaga od dwóch do trzydziestu cykli zegarowych.

Projektując mikroprocesor mający konkurować z minikomputerami, konstruktorzy musieli zwiększyć obszar bezpośrednio dostępnej pamięci (tzn. takiej, którą mikroprocesor może zaadresować bez potrzeby przełączania sprzętowo zrealizowanych stron). Mikroprocesor 8086 może bezpośrednio wykorzystywać do 1 megabajta pamięci.



Rys. 2. Metoda wyznaczania adresu fizycznego przy operacji dostępu do pamięci

a) rejestry, które mogą być uwzględnione w operacjach na danych

b) przykład wyznaczania adresu z uwzględnieniem wartości podanej bezpośrednio w kodzie programu

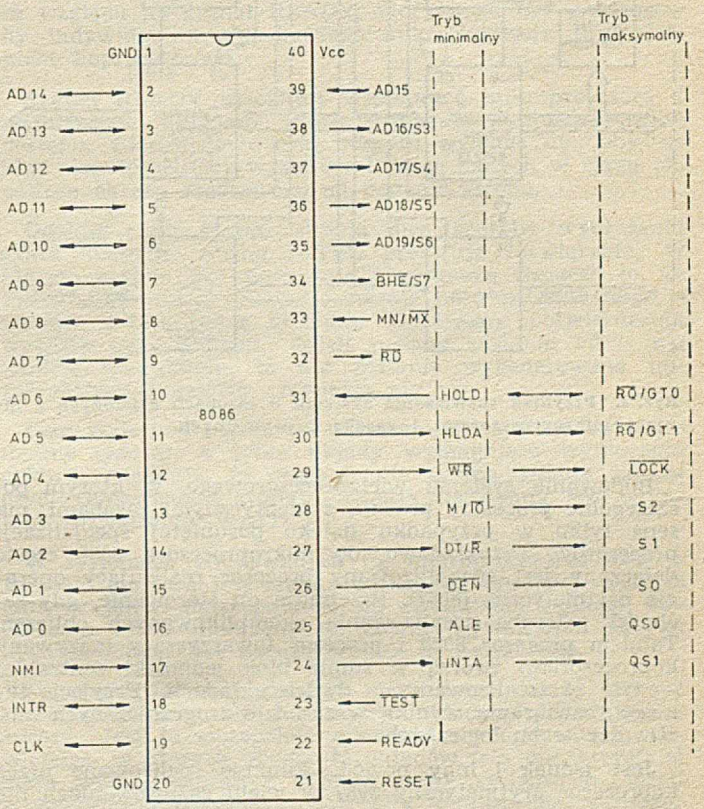
Wszelkie operacje dostępu do pamięci odbywają się za pośrednictwem rejestru segmentu pamięci (rys. 2). Dzięki takiemu rozwiązaniu wszystkie programy są relokowalne. Oznacza to, że kod programu można przepisywać w dowolne miejsca pamięci i zlecać jego realizację CPU, bez potrzeby wprowadzania modyfikacji.

Możliwe jest także wykorzystywanie trybu adresowego, w którym adresy fizyczne obliczane są z uwzględnieniem bieżącej wartości rejestru SI. Napisane w ten sposób procedury wielobieżne (ang. reentrant) mogą zostać ponownie wywołane przez wstrzymujące ich realizację zewnętrzne przerwanie bez obawy zamazania wcześniej zapisanych informacji. Procedura może też wywoływać samą siebie.

Rozbudowane metody generowania fizycznych adresów pamięci są świetnym narzędziem w rękach doświadczonego programisty. Równocześnie jednak tworzą trudną do sforsowania barierę dla mniej wprawnych osób. Dlatego też znaczna część programistów preferuje stosowanie języków wyższego poziomu. Sprzyja to zwiększeniu przejrzystości pisanych przez nich programów, a także sprawia, że finalny produkt często ma efektywność działania zbliżoną do programów pisanych w języku ASSEMBLER dla mikroprocesorów 8-bitowych. Można stwierdzić, że większa moc przetwarzania mikroprocesora 8086 jest wykorzystywana w dużej mierze dla polepszenia komfortu pracy programistów.

Mikroprocesor 8086 może działać w dwóch różnych trybach: minimalnym oraz maksymalnym (rys. 3). Wybór trybu pracy nie ma wpływu na realizację programu, zmienia natomiast sposób współpracy z otoczeniem. Tryb wybierany jest przez przyłączenie wyprowadzenia MN-MX/ do masy (tryb maksymalny) lub +5 V (tryb minimalny).

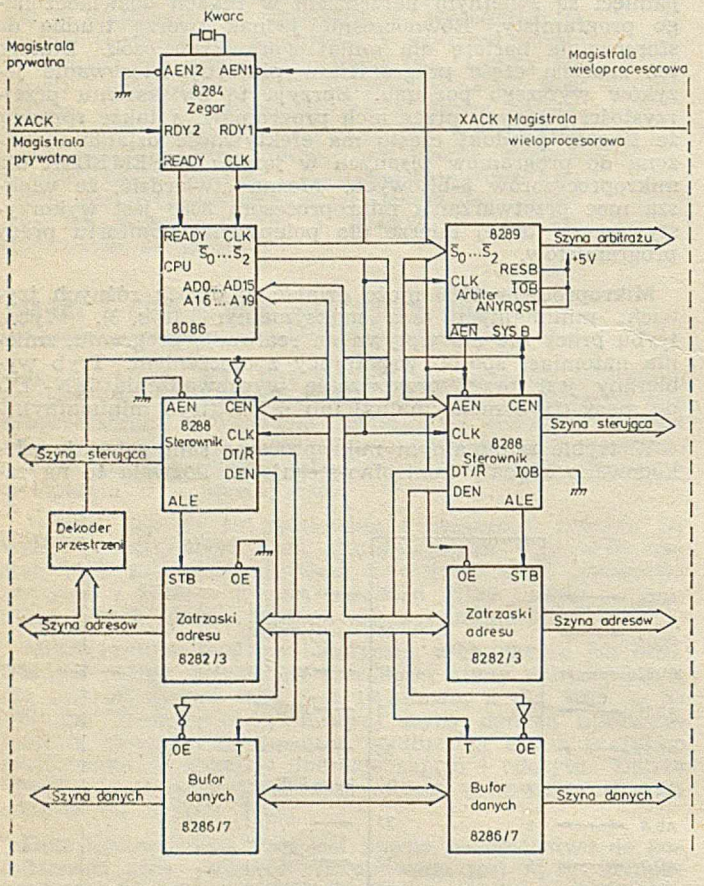
W trybie minimalnym mikroprocesor sam generuje zdekodowane sygnały kontrolno-sterujące. Pozwala to na za-



Rys. 3. Topografia wyprowadzeń układu 8086 z uwzględnieniem dwóch trybów pracy: minimalnego i maksymalnego

oszczędzenie dodatkowego układu sterującego szyną. Wykorzystanie trybu minimalnego ogranicza się do systemów jednoprocesorowych. W trybie tym jedynie układ DMA może na pewien czas przejąć sterowanie magistralami.

W trybie maksymalnym 8086 generuje jedynie sygnały statusu informujące o typie wykonywanej operacji. Dzięki temu możliwe jest zastosowanie dwóch (lub więcej) układów sterujących różnymi szynami. W ten sposób pamięć i układy wejścia-wyjścia można rozdzielić na zasoby prywatne mikroprocesora i zasoby dostępne również dla innych mikroprocesorów pracujących w systemie (rys. 4). Możliwość wprowadzenia takiego podziału w systemie wieloprocesorowym jest podstawowym warunkiem efektywnej pracy, gdyż w danym momencie z magistrali systemowej może korzystać tylko jeden procesor. Pozostałe muszą czekać — chyba że dysponują prywatną pamięcią, z której odczytują program bez wchodzenia w kolizję z mikroprocesorem wykorzystującym magistralę systemową.



Rys. 4. Przykład rozwiązania systemu o zasobach dzielonych z innymi mikroprocesorami i zasobach prywatnych

Budowanie systemu wieloprocesorowego, w którym poszczególne procesory pracują z identycznymi zasobami, ma sens tylko w przypadku daleko posuniętej specjalizacji procesorów. Przykładowo, do mikroprocesora 8086 często dołączany jest specjalizowany procesor realizujący operacje arytmetyczne (8087). Przejmuje on sterowanie, gdy zachodzi potrzeba zrealizowania skomplikowanych obliczeń. Tandem procesor 8086 i procesor towarzyszący (nazywany koprocesorem) tworzą w sumie blok jednostki centralnej — tyle że zrealizowany na dwóch układach. Przyjęcie takiego rozwiązania wynika w zasadzie z ograniczonych możliwości technologicznych.

Jest jednak i inny powód. Funkcje realizowane przez koprocesor arytmetyczny 8087 w wielu zastosowaniach nie są potrzebne, natomiast przydatne mogłyby się okazać zupełnie inne. Wychodząc naprzeciw różnym potrzebom użytkowników firma INTEL przygotowuje całą rodzinę specjalizowanych koprocesorów. Obecnie oferowany jest już koprocesor LAN 82586 (ang. Local Area Network — układ realizujący funkcje związane z obsługą sieci lokalnych), koprocesor tekstu 82730 (dedykowany zagadnieniom związanym z przetwarzaniem plików tekstowych) i koprocesor wejścia-wyjścia 8089. Ostatni z wymienionych może być również wykorzystywany jako samodzielny procesor specjalizujących się w realizacji funkcji związanych z obsługą kanałów wejścia-wyjścia.

Dołączenie jednego z koprocesorów nie wyklucza zastosowania następnego. Możliwe jest dzięki temu komponowanie jednostek centralnych o dużej mocy przetwarzania, dostosowanych do obsługi konkretnego zagadnienia.

W trybie mnimalnym dla przejęcia sterowania przez układ DMA wykorzystywane są sygnały HOLD i HLDA. W trybie maksymalnym założono, że sterowanie będzie przejmowane jedynie przez koprocesory. Dlatego też odstąpiono od tradycyjnego protokołu wprowadzając w zamian wymianę żądania i potwierdzenia (RQ/—GT/) przesyłane po tej samej linii. Układ, który chce przejąć sterowanie, zgłasza ten zamiar generując impuls RQ/. Z chwilą zakończenia aktualnie wykonywanej operacji, 8086 generuje impuls potwierdzenia GT/ i zwalnia szynę.

Przyjęte rozwiązanie umożliwia prostą realizację współpracy nawet z większą liczbą koprocesorów (dwie linie o różnych priorytetach, możliwość kaskadowego przekazywania żądań i potwierdzeń). Natomiast dołączenie tradycyjnego układu DMA (np. 8257, 8237) do działającego w trybie maksymalnym 8086 wymaga zrealizowania stosunkowo złożonej konstrukcji układowej.

W trybie maksymalnym możliwy jest odczyt statusu kolejki rozkazów (wyprowadzenia QS0 i QS1). Pozwala to wykorzystać zewnętrzne urządzenia do śledzenia pracy mikroprocesora. Informacja o stanie kolejki wykorzystywana jest również przez koprocesory rozszerzające zestaw rozkazów CPU.

Należy jeszcze wspomnieć o sygnale LOCK/ generowanym tylko w trybie maksymalnym. Blokuje on możliwość rozdzielenia realizacji poszczególnych cykli rozkazu przez inny procesor. Jest to o tyle istotne, że w systemach wieloprocesorowych po każdym cyklu magistrali rozpatrywane są aktualne zgłoszenia i sterowanie przejmuje mikroprocesor o najwyższym priorytecie. Przerwanie rozkazu typu odczyt-modyfikacja-zapis mogłoby spowodować błąd.

Układ 8086 umieszczony został w obudowie 40-wyprowadzeniowej. Z tego powodu konieczne było wprowadzenie multipleksowanej szyny danych i adresowej. W okresie T1 operacji dostępu na szynie pojawia się adres (pamięci lub portu). W okresie T2 musi on ustąpić miejsca danym. Tak więc wystawiony na szynę adres trzeba zapamiętać w zewnętrznych rejestrach, które sterują blokiem dekodera. Możliwości wykorzystania informacji adresowej bez stosowania rejestrów zatrzaskowych są raczej ograniczone.

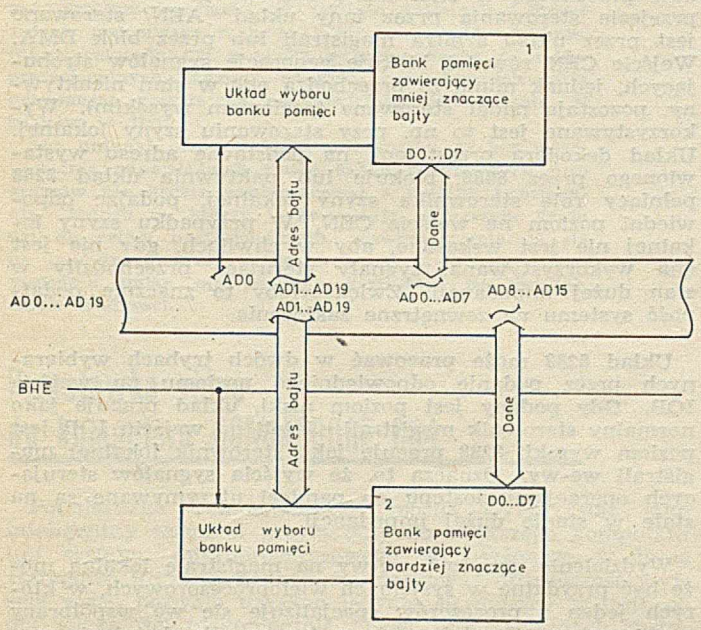
Multipleksowana szyna adresów i danych stwarza większe trudności w uruchamianiu i testowaniu systemów. „Zamrożenie" wykonywanej operacji jest możliwe tylko w cyklu T3 (precyzyjniej: powtarzana jest realizacja cyklu identycznego z T3), a więc odczyt wystawionego wcześniej adresu można zrealizować wyłącznie za pośrednictwem rejestrów pamiętających. W przypadku mikroprocesora 8086 nie można zatrzymać zegara, gdyż w strukturze wykorzystano komórki pamięci dynamicznych.

Pojawiający się na multipleksowanej szynie adres może być zapamiętany w zewnętrznych rejestrach przy wykorzystaniu sygnału ALE (ang. Address Latch Enable). Przechodzi on w stan aktywny 1, gdy na szynę podany został adres. Opadające zbocze ALE powinno zatrzaskiwać adres w rejestrze.

Dodatkową trudnością wynikającą z multipleksowanej szyny adresów i danych jest konieczność zagwarantowania, że dane z pamięci lub układu wejścia-wyjścia nie będą zakłócać wystawianych adresów. Dlatego obwody wyjściowe współpracujących układów muszą być uaktywniane nie przez sygnał z dekodera adresów, lecz przez

strob odczytu MRDC/ lub IORC/. Układy produkowane przez firmę INTEL mają niezależne wejścia CS/ i RD/, co eliminuje możliwość konfliktu.

Mikroprocesor 8088 przesyła dane jedynie w postaci bajtowej. 8086 przesyła słowa (16 bitów) oraz bajty. Przesłania bajtowe wprowadzono dlatego, że kody niektórych rozkazów składają się z nieparzystej liczby bajtów. W ten sposób programy zajmują mniej miejsca, lecz organizacja pamięci jest bardziej złożona, przez co komplikują się również układy dekoderów. Przyjęte rozwiązanie wydaje się obecnie dość kontrowersyjne. Układy pamięci są już stosunkowo tanie, natomiast oszczędność miejsca jest raczej niewielka.



Rys. 5. Organizacja pamięci w systemie sterowanym przez mikroprocesor 8086

Na rysunku 5 przedstawiono sposób organizacji pamięci. Składa się ona z dwóch banków o organizacji bajtowej. Pierwszy bank przyłączony jest do mniej znaczącego bajtu szyny danych i wybierany jest wtedy, gdy na linię adresową AD0 podany zostanie stan logiczny 0. Drugi bank wybierany jest przez poziom aktywny na linii BHE/ (ang. Bus High Enable) i przyłączony jest do bardziej znaczącego bajtu szyny danych. Odczyt słowa (16-bitowy) następuje tylko w przypadku równoczesnego wybrania obydwu banków. Poszczególne komórki w ramach banków pamięci wybierane są stanami logicznymi na liniach AD1 ... AD15 i A16 ... A19. Tak więc każdy z banków może pomieścić do 512 KB pamięci.

W przyjętym rozwiązaniu odczyt słowa może nastąpić jedynie spod parzystego adresu (tylko wtedy AD0 przyjmuje stan 0). Jeżeli wykonywany rozkaz wymaga dostępu do słowa umieszczonego pod nieparzystym adresem, to CPU automatycznie wykonuje dwa cykle: najpierw dostęp do bajtu umieszczonego w banku 2, a następnie dostęp do bajtu umieszczonego w banku 1. Równoczesny dostęp do obydwu banków nie jest w tym przypadku możliwy, gdyż komórki w poszczególnych bankach różnią się stanem logicznym na linii AD1.

Jeżeli realizowany jest dostęp do bajtu (a nie słowa) umieszczonego pod parzystym adresem, linia BHE/ przyjmuje stan pasywny 1. Podobnie, przy dostępie do bajtu umieszczonego pod nieparzystym adresem, linia AD0 jest w stanie 1. Warto zwrócić uwagę, że przy ładowaniu słowa umieszczonego pod nieparzystym adresem do rejestru 16-bitowego wykonane zostaną dwa cykle odczytu. Najpierw wprowadzony zostanie bajt z banku dołączonego do linii AD8 ... AD15 szyny danych. Przeznaczony jest on dla mniej znaczącego bajtu rejestru 16-bitowego. Drugi odczyt nastąpi z banku przyłączonego do lini AD0 ... AD7 i trafi do bardziej znaczącego bajtu rejestru.

Ten pozornie zagmatwany sposób dostępu staje się bardziej przejrzysty, jeżeli linię adresową AD0 potraktować podobnie jak BHE/ — czyli strob wyboru banku. Wtedy tylko AD1 ... AD15 są liniami wyznaczającymi adres komórki. Poszczególne bajty składające się w słowa umieszczane są na przemian w obydwu bankach.

Linie adresowe A16 ... A19 nie są multipleksowane z danymi, lecz z informacjami o charakterze statusu. A16 i A17 informują o segmencie bazowym, który wykorzystywany jest w aktualnie realizowanej operacji. Linia A18 odzwierciedla stan wskaźnika informującego, czy możliwe jest przyjmowanie przerwań. Linia A19 utrzymywana jest w stanie 0, gdy mikroprocesor steruje szyną.

Dość specyficzną rolę spełnia wyprowadzenie o nazwie TEST/. Jest ono wykorzystywane w połączeniu z rozkazem WAIT, który wprowadza CPU w stan oczekiwania na pojawienie się poziomu aktywnego na wejściu TEST/. Podobną operację można by w zasadzie zrealizować wykorzystując przerwania. Jednak wejście TEST/ można wykorzystać także wtedy, gdy przerwania są zablokowane lub gdy 8086 przekazał sterowanie do koprocesora (znajdujące się w kolejce rozkazy realizowane są dopóki BIU nie otrzyma zlecenia wymagającego dostępu poprzez szynę).

UKŁAD ZEGARA 8224

Konstrukcja oparta na mikroprocesorze 8086 musi zawierać, podobnie jak w przypadku mikroprocesora 8080, dodatkowy układ zegara. Układ 8284, poza generacją sygnału zegarowego taktującego pracą mikroprocesora, wykorzystywany jest do synchronizacji niektórych sygnałów kontrolnych (np. RESET, READY).

W systemach wieloprocesorowych mogłoby wydawać się celowe zastosowanie jednego układu zegara 8284 dla wszystkich mikroprocesorów. Nie jest to jednak możliwe ze względu na sygnał READY, który musi być generowany indywidualnie dla każdego mikroprocesora (z wyjątkiem koprocesorów).

Sygnał READY umożliwa współpracę mikroprocesora z wolniejszymi układami, które wymagają wydłużenia cyklu dostępu. Standardowy cykl odczytu wymaga np., aby układ współpracujący wysterował szynę danych w ciągu ok. półtora okresu zegarowego od podania strobu.

Ogólnie rzecz biorąc istnieją dwa sposoby wydłużania cyklu operacji. Firma INTEL preferuje rozwiązanie, w którym system jest normalnie niegotowy. Oznacza to, że każdy zaadresowany układ generuje potwierdzenie umożliwiające dalszą pracę systemu. Opóźnienie potwierdzenia wydłuża cykl dostępu. Jeżeli podany zostanie adres nie istniejącego układu, to nie zostanie wygenerowane potwierdzenie i system „zawiesza się".

Inne podejście polega na założeniu, że system jest normalnie gotowy, a tylko układy wymagające wydłużenia cyklu sygnalizują, że operacji nie można jeszcze zakończyć. Odpowiednik sygnału READY nazywany jest w takim rozwiązaniu WAIT/.

W systemach zbudowanych na mikroprocesorze 8086 w zasadzie możliwe jest zrealizowanie obu koncepcji. Jednak czas, w jakim musi pojawić się ewentualne żądanie wydłużenia cyklu (mierzony od chwili wystawienia aktywnych strobów), jest bardzo trudny do osiągnięcia.

Do wad systemu normalnie niegotowego należy zaliczyć konieczność znacznej rozbudowy układów dekodujących adresy. Jest to szczególnie odczuwalne w systemach niewielkich, w których większość układów (ale nie wszystkie) może współpracować z CPU z pełną szybkością. Jeżeli z określonych powodów nie zostały umieszczone w zbliżonym obszarze przestrzeni adresowej „gotowe" układy, to każdy z nich należy rozbudować o blok generacji potwierdzenia READY.

Jeżeli sygnał READY generowany jest po szczegółowym określeniu komórki przez dekoder, to może on dotrzeć do CPU zbyt późno i cykl dostępu zostaje niepotrzebnie wydłużony. Dlatego też niedawno wprowadzono nową wersję układu zegarowego 8284A. Umożliwia on wykorzystanie zewnętrznych układów TTL do synchronizacji READY z zegarem sterującym pracą CPU.

Dla ochrony przed skutkami zablokowania systemu przez zaadresowanie nieistniejącej komórki, warto wprowadzić specjalny układ czasowy zwany budzikiem (ang. watch dog timer), który po zbyt długim braku potwierdzenia generuje je sam, sygnalizując ewentualnie błędną sytuację zgłoszeniem przerwania.

Systemy normalnie gotowe wymagają bardzo szybkiego stwierdzenia, że zaadresowany został układ wymagający wydłużenia cyklu. Utrudnia to realizację rozbudowanych systemów, szczególnie systemów wieloprocesorowych.

Podsumowując, 8086 predestynowany jest do pracy w rozbudowanych systemach skonstruowanych jako normalnie niegotowe lub w systemach, gdzie nie zachodzi potrzeba wydłużania cyklu operacji (w tym przypadku system jest stale gotowy). Również praca 8086 w trybie minimalnym nie zmienia powyższej konkluzji, gdyż czas pozostawiony na ewentualne zgłoszenie WAIT/ jest jeszcze krótszy. Jest to wyraźna wada, gdyż tryb minimalny miał upraszczać konstruowanie niewielkich systemów.

READY powstaje ze zsumowania sygnałów z wielu źródeł. Dla uproszczenia rozwiązań układowych w 8284 przewidziano cztery wejścia dla obsługi tego zgłoszenia. Możliwe jest sumowanie sygnałów z dwóch źródeł, przy czym w zależności od polaryzacji sygnału można wykorzystać albo wejście AEN/ albo RDY. Wykorzystanie dwóch wejść dla danego źródła istotne jest w przypadku współpracy z magistralą systemu wieloprocesorowego dla odcięcia potwierdzeń przychodzących w czasie, gdy procesor nie steruje tą magistralą. Najczęściej zbuforowana inwerterem Schmitta systemowa linia potwierdzeń XACK/ doprowadzona jest do wejścia RDY. Na wejście AEN/ podawany jest sygnał z układu arbitra magistrali (8289) określający, czy dany mikroprocesor uzyskał dostęp do magistrali systemowej.

Podstawowym zadaniem układu 8284 jest generacja sygnału zegarowego sterującego mikroprocesorem. Jego częstotliwość musi być większa od 2 MHz. Maksymalna częstotliwość zależy od wersji mikroprocesora — standardowo wynosi ona 5 MHz. Układ może generować sygnał zegarowy w oparciu o przyłączony kwarc lub sygnał zewnętrzny (zarówno kwarc, jak i przebieg zewnętrzny muszą mieć częstotliwość trzykrotnie większą od zamierzonej częstotliwości dla 8086). Wykorzystanie zewnętrznego źródła może być przydatne do synchronizacji przebiegów zegarowych w systemach wieloprocesorowych. Przesunięcia poszczególnych sygnałów zegarowych nie powinny wtedy przekroczyć 45 ns.

Układ 8284 zawiera wewnętrzny dzielnik częstotliwości przez trzy, którego stan po włączeniu zasilania nie jest zdeterminowany. Z tego powodu wprowadzono dodatkowe wejście synchronizujące CSYNC, umożliwiające zainicjowanie pracy zegara przez zewnętrzny impuls.

Układ 8284 ma trzy wyjścia sygnałów zegarowych. Wyjście OSC generuje przebieg o częstotliwości podłączonego kwarcu i może być wykorzystane jako źródło sygnału synchronizującego inne układy 8284. Wyjście CLK dostarcza przebieg sterujący mikroprocesorem. Wreszcie wyjście PCLK generuje przebieg o częstotliwości dwukrotnie mniejszej od CLK. Zegar ten wykorzystywany jest zazwyczaj przez pracujące w systemie liczniki.

UKŁAD STEROWNIKA SZYNY 8288

Układ sterownika szyny 8288 wykorzystywany jest w systemach, w których 8086 działa w trybie maksymalnym (MN-MX/ przyłączone jest do masy). Zamienia on sygnały statusu S0, S1, S2 na sygnały strobów sterujących wykonywanymi w systemie operacjami. Tak proste zadanie spełniłby jednak z równym powodzeniem zwyczajny dekoder: 1 z 8. Sterownik 8288 realizuje wiele dodatkowych funkcji.

Najistotniejszą korzyścią wynikającą z zastosowania układu 8288 jest możliwość pracy w systemach o wielu magistralach. W tym celu przewidziano wejścia kontrolne AEN/ i CEN. AEN/ wykorzystywane jest przy współpracy z magistralą wieloprocesorową lub z DMA. Jeżeli na wejście AEN/ podany zostanie poziom wysoki, sygnały strobów przechodzą w stan dużej impedancji. Umożliwia to przejęcie sterowania przez inny układ. AEN/ sterowane jest przez układ arbitra magistrali lub przez blok DMA. Wejście CEN również blokuje generację sygnałów strobujących, jednak mimo że przechodzą one w stan nieaktywny, pozostają nadal sterowane (poziomem wysokim). Wykorzystywane jest to np. przy sterowaniu szyny lokalnej. Układ dekodera przestrzeni, na podstawie adresu wystawionego przez 8086, blokuje lub uaktywnia układ 8288 pełniący rolę sterownika szyny lokalnej, podając odpowiedni poziom na wejście CEN. W przypadku szyny lokalnej nie jest wskazane, aby w chwilach, gdy nie jest ona wykorzystywana, sygnały sterujące przechodziły w stan dużej impedancji. Zwiększyłoby to znacznie podatność systemu na zewnętrzne zakłócenia.

Układ 8288 może pracować w dwóch trybach wybieranych przez podanie odpowiedniego poziomu na wejście IOB. Gdy podany jest poziom niski, układ pracuje jako normalny sterownik magistrali. Jeżeli na wejściu IOB jest poziom wysoki, 8288 pracuje jako sterownik lokalnej magistrali we-wy. Oznacza to, że wyjścia sygnałów sterujących operacjami dostępu do pamięci utrzymywane są na stałe w stanie dużej impedancji.

Wydzielenie urządzeń we-wy na magistralę lokalną może być przydatne w systemach wieloprocesorowych, w których jeden z procesorów specjalizuje się we współpracy z otoczeniem. Zmniejsza się w ten sposób obciążenie magistrali systemowej. Jednak korzyści płynące z wykorzystania do tego celu specyficznego trybu pracy sterownika 8288 wydają się niewielkie. Przykładowo, nie można całkowicie zrezygnować z dekodera przestrzeni, bowiem nie przewidziano dla układu 8288 trybu pracy, w którym sterowałby on tylko operacjami związanymi z pamięcią. Niezbędne jest więc zastosowanie układu dekodującego rodzaj realizowanej operacji i ewentualnie blokującego sterownik obsługujący magistralę systemową.

Układ 8288 rozszerza zestaw sygnałów sterujących o przyspieszone (ang. advanced) stroby zapisu. Pojawiają się one w tym samym cyklu zegarowym, co stroby odczytu, a więc o ok. 1 cykl wcześniej niż standardowe stroby zapisu. Przyspieszone stroby zapisu wygodnie jest wykorzystać w układach generujących potwierdzenie operacji (sygnał READY lub XACK) dla uniknięcia zbędnych cykli oczekiwania. Nie należy ich jednak wykorzystywać do sterowania buforów szyny danych, gdyż przechodzą one w stan aktywny zanim CPU zdąży wystawić dane.

Generowany przez 8288 sygnał MCE stosowany jest w systemach o rozbudowanym bloku obsługi przerwań, wykorzystującym możliwość kaskadowego łączenia układów 8259A. W proponowanym przez producenta rozwiązaniu strob MCE otwiera przejście dla adresu generowanego przez zarządzający układ przerwań. Adres ten pozwala podporządkowanym sterownikom przerwań zorientować się, czyje żądanie zostało spełnione.

ANDRZEJ J. PIOTROWSKI

Poniższy materiał został zainspirowany przystawką opisaną w amerykańskim czasopiśmie BYTE. Zamawiając materiał poleciliśmy jednak autorowi przystosowanie konstrukcji do krajowych warunków — oczywiście tam, gdzie ma to sens. Dostrzegamy też dla opisanej konstrukcji nieco inną klasę zastosowań niż autorzy oryginalnego rozwiązania. Poszukiwali oni taniego urządzenia typu terminal, które mogliby wykorzystać do współpracy z mikrokomputerami budowanymi w oparciu o mikroprocesor 68000. Relatywnie niska cena mikrokomputera C64 w USA skłoniła ich do przyjęcia takiego właśnie rozwiązania.

Rozpatrując dostępność i cenę C64 w Polsce nie sądzimy, aby ktokolwiek w ten sposób próbował rozwiązać problem trudności w zakupie terminali. Biorąc jednak pod uwagę fakt, że wielu krajowych użytkowników zakupiło C64 z myślą o zastosowaniu do celów profesjonalnych, sądzimy, że nieco urealnimy ich wizje proponując przystawkę umożliwiającą wyświetlanie 80 znaków w linii.

Aby uniknąć nieporozumień zaznaczamy, że bez wprowadzania odpowiednich zmian żaden z firmowych programów nie będzie współpracował z opisaną konstrukcją. Jednak ci, którzy wykorzystują w naszym kraju C64 do celów profesjonalnych, zazwyczaj opierają się na napisanych przez siebie programach, tak że wprowadzenie zmian nie powinno nastręczyć im zasadniczych trudności.

Przystawka video do komputera COMMODORE C64

Popularnego Commodora C64 można wykorzystać jako inteligentny terminal przyłączany do większego komputera lub systemu mikroprocesorowego wyposażonego w łącze szeregowe RS 232C. Jednak standardowa rozdzielczość wyjścia video pozwala uzyskać jedynie 40 znaków w linii, co dla większości niekoniecznie nawet profesjonalnych zastosowań jest stanowczo za mało. Rozwiązaniem tego problemu może być specjalna przystawka, która umożliwi wyświetlenie 80 znaków w jednej linii. Przyłączana jest ona do portu rozszerzającego Commodora (ang. Commodore Expansion Port) i zawiera: CRTC 6845, 2KB SRAM (pamięć obrazu) i 4KB EPROM (pamięć nakładki programowej i generator znaków). Idea przystawki została przedstawiona na schemacie blokowym (rys. 1).

Funkcjonowanie przystawki

Układ CRTC generuje adresy wyświetlanej na ekranie komórki pamięci obrazowej na liniach MA0 ... MA10. Adres kolejnego rzędu wyświetlanego znaku podawany jest na liniach RA0 ... RA3. W poszczególnych komórkach pamięci obrazowej zawarte są kody ASCII (American Standard Code for Information Interchange) znaków, które mają być wyświetlone na ekranie. Odczytywany z pamięci obrazowej kod znaku wykorzystywany jest do zaadresowania pamięci w generatorze znaków. Ponieważ każdy znak składa się z 8 rzędów kropek, to dla wybrania odpowiedniej komórki w pamięci generatora znaków niezbędne jest jeszcze dołączenie adresu aktualnie wyświetlanego na ekranie rzędu.

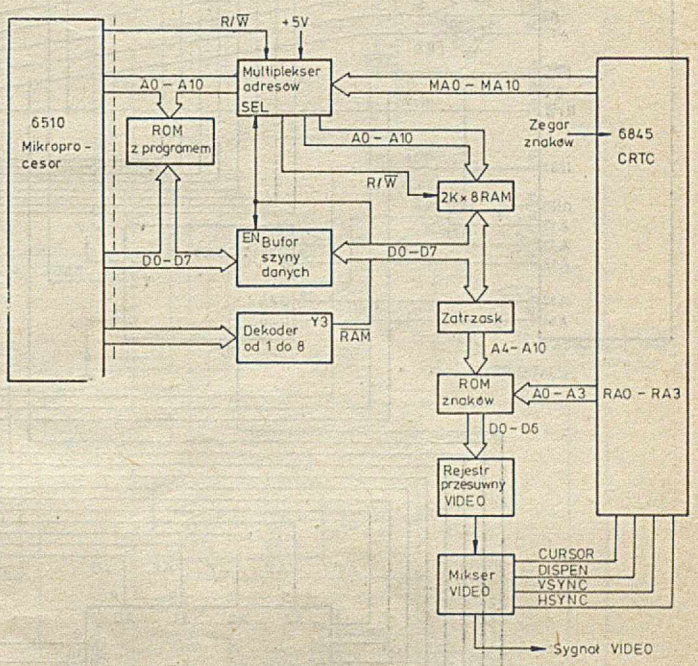
Kody odczytywane z kolejnych komórek pamięci obrazu zatrzaskiwane są na liniach adresowych pamięci ROM generatora znaków. Z CRTC podawany jest dodatkowo adres związany z aktualnie wyświetlanym rzędem. Kombinacja kodu ASCII i adresu rzędu formuje adres komórki w pamięci ROM generatora znaków, w której znajduje się odzwierciedlenie odpowiedniego rzędu dla aktualnie wyświetlanej litery. Poszczególne bity kodu znajdującego się w pamięci ROM określają, czy kolejne punkty na ekranie mają być rozjaśnione czy zaciemnione.

CRTC rozpoczyna od sekwencyjnego generowania adresów pamięci obrazowej odpowiadających 80 komórkom dla znaków z pierwszego wiersza. W tym czasie na linie adresowe rzędu podawany jest stan 0 — co oznacza, że wyświetlany jest rząd zerowy początkowych 80 znaków. Po poziomym powrocie plamki (koniec wyświetlania pierwszej linii obrazu) CRTC ponownie generuje tę samą sekwencję 80 adresów pamięci obrazu, ale tym razem z adresem rzędu równym 1 (druga linia obrazu). Cykl ten jest powtarzany do chwili aż zostaną wyświetlone wszystkie osiem rzędów dla 80 znaków z pierwszego wiersza tekstu. Następny blok 80 znaków jest generowany z adresem rzędu ustawionym znowu na zero, potem na 1 itd. Cała procedura kontynuowana jest aż do wypełnienia ekranu. Po pionowym powrocie plamki proces jest powtarzany.

Informacja z pamięci ROM generatora znaków przepisywana jest do rejestru przesuwanego, z którego sekwencyjnie (bit po bicie) podawana jest do bloku miksera video. Zadaniem miksera jest utworzenie zespolonego sygnału wizyjnego.

Na rysunku 1 pokazano również interfejs między płytką video a Commodore C64. Zarówno komputer, jak i sterownik CRT, muszą mieć dostęp do 2KB pamięci obrazowej. Układ mikroprocesora 6510 wpisuje do niej kody poszczególnych znaków ASCII, a układ 6845 steruje przesyłaniem jej zawartości do wyświetlania. W proponowanym rozwiązaniu 2KB pamięci obrazu umieszczono między adresami 9800_H ... $9FEF_H$.

Dostępem do pamięci zarządza multiplekser sterowany przez układ procesora 6510. Wystawienie przez mikroprocesor 6510 adresu z obszaru pamięci obrazu powoduje, że na wyjściu dekodera Y_3 (RAM/) pojawia się stan aktywny (logiczne zero). Sygnał ten otwiera bufor szyny danych oraz przełącza multiplekser adresowy na przekazywanie adresów i sygnału R-W/ pochodzących od mikroprocesora. Gdy 6510 nie wykonuje operacji związanej z pamięcią obrazu, adresy są wybierane przez układ 6845, a na wejście R-W/ pamięci obrazowej multiplekser dołącza sygnał +5V. W ten sposób pamięć jest utrzymywana przez cały ten okres w trybie odczytu.



Rys. 1. Schemat blokowy przystawki video do COMMODORE C 64

Układ taktujący (rys. 2) jest sercem obwodu video. Steruje on przesyłaniem wszystkich danych. Kwarc 16 MHz, trzy inwertery 7404 i dwa oporniki 620 Ω składają się na generator sygnału DOT — przebiegu prostokątnego o częstotliwości 16 MHz. Jest on wykorzystywany przez licznik 74161, czterokrotny przerzutnik typu D 74174 oraz in-

wertery do wysterowania sygnałów taktujących systemem: S-L/, CC i CC/. Gdy S-L/ jest sterowany poziomem wysokim, rejestr przesuwny przesyła dane do miksera video zgodnie z sygnałem taktującym DOT. Gdy S-L/ jest sterowany poziomem niskim, do rejestru przesuwnego wpisywane są nowe dane oraz przepisywany jest z pamięci obrazowej do układów zatrzaskowych 74174 nowy znak kodu ASCII.

Sygnał CC pełni rolę zegara znaków. Impuls generowany jest co dziewięć cykli zegara DOT. Jest to czas niezbędny do wyświetlenia jednej linii jednego znaku. Zegar znaków synchronizuje pracę sterownika CRT z resztą systemu. Sygnał CC/ jest sygnałem odwrotnym do zegara znaków. W generatorze znaków część obwodu składająca się z układów 74174 jest używana do zatrzymywania danych na liniach adresowych doprowadzonych do pamięci ROM. Opadające zbocze sygnału zegara znaków przygotowuje linie adresowe MA0—MA10. Ponieważ informacja na wyjściu pamięci obrazowej może się jeszcze zmienić, to dopiero narastające zbocze sygnału S-L/ powoduje zatrzaśnięcie danych. Ten sam sygnał wyzwala rejestr przesuwny 74165, co powoduje wysłanie kolejnych bitów poprzedniego znaku do miksera video. Przesyłanie danych z pamięci obrazowej do generatora znaków, a następnie do rejestru przesuwnego, powoduje opóźnienie o dwa cykle zegara znaków (od momentu zaadresowania pamięci obrazowej do momentu, gdy dana opuszcza rejestr przesuwny).

W obwodzie miksera video sygnały VIDEO, CURSOR, DISPEN, S-L/, HSYNC i VSYNC łączą się tworząc zespolony sygnał wizyjny.

Sygnały CURSOR i DISPEN z układu CRTC są przy użyciu dwóch układów 74174 opóźniane o dwa cykle zegara znaków, tak aby skompensować zwłokę dwóch znaków, o której była mowa wcześniej. Układ 74174 jest taktowany odwróconym zegarem znaków (CC/), ponieważ sygnały na jego wyjściach są zmieniane przez narastające zbocze zegara, odwrotnie niż w CRTC, którego wyjścia zmieniają się przy opadającym zboczu. Opóźnione sygnały CURSOR i DISPEN wraz z sygnałem S-L/, zmieszane z sygnałem VIDEO, tworzą sygnał bezpośrednio sterujący jasnością plamki.

Przesunięcia między wspomnianymi sygnałami a sygnałem VIDEO powodują, że sumaryczny sygnał jest nieco „zaśmiecony". Sterowany sygnałem DOT przerzutnik typu D wydziela prawidłowy sygnał jaskrawości. Jest on następnie mieszany przez tranzystor BD 140 z sygnałami HSYNC i VSYNC, co w sumie daje zespolony sygnał wizyjny. Tranzystor BD 139 pracujący w układzie wtórnika emiterowego dostosowuje ten sygnał do 75-omowego wejścia monitora.

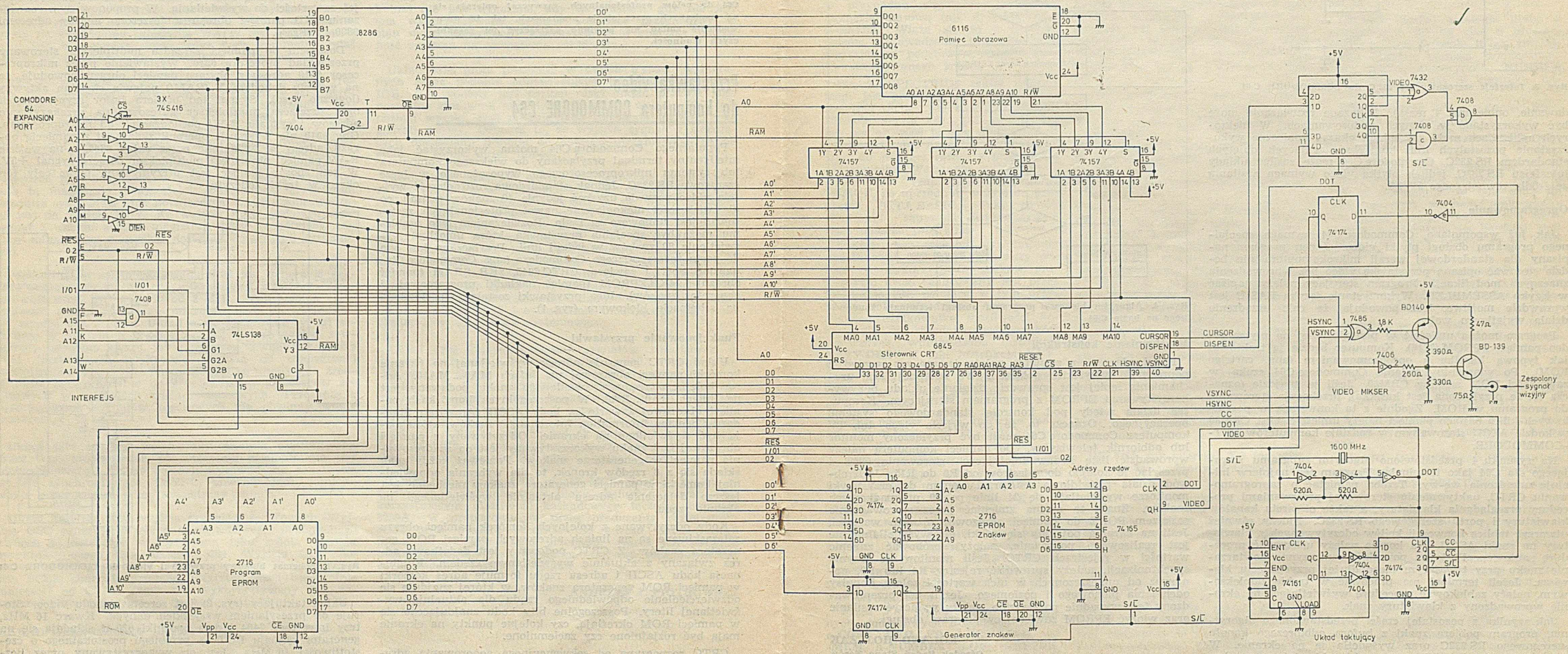
W bloku sterowania dostępem do pamięci trzy układy 74157 (czterowejściowe selektory linii danych) tworzą multiplekser adresów. Linie A11 ... A15 z mikroprocesora podawane są na układ 74LS138 (dekoder 1 z 8), który generuje sygnał RAM/. Gdy mikroprocesor generuje dostęp do pamięci obrazu, sygnał RAM/ przechodzi w stan 0, co powoduje wybór wejść A układów 74157. Gdy RAM/ przechodzi w stan 1, mikroprocesor nie korzysta z pamięci obrazu i wybierane są wejścia B przekazujące sygnały z linii adresowych MA0 ... MA10 do pamięci obrazu. Sygnał RAM/ doprowadzony jest również do układu 8286, co powoduje przyłączenie szyny danych do pamięci RAM wtedy, gdy mikroprocesor realizuje dostęp do pamięci obrazowej.

Jeżeli pamięć obrazu nie jest adresowana przez mikroprocesor, to bufory przechodzą w stan trzeci, izolując w ten sposób szynę 6510 od operacji realizowanych przez CRTC.

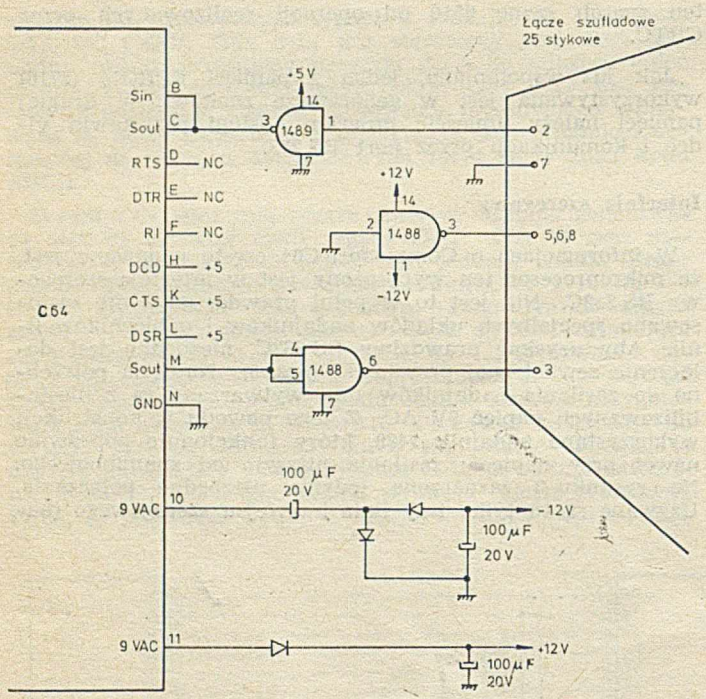
Jak już wspomniano, jedna z pamięci EPROM (2716) wykorzystywana jest w generatorze znaków. W drugiej pamięci należy umieścić program obsługi przystawki video i komunikacji przez port RS 232.

Interfejs szeregowy

W informacjach o Commodore C64 często podawane jest, że mikroprocesor ten wyposażony jest w interfejs szeregowy RS 232C. Nie jest to w pełni prawda, gdyż nie zastosowano specjalnych układów nadajników i odbiorników linii. Aby uzyskać prawdziwy RS 232C, niezbędne jest dołączenie zewnętrznej przystawki (rys. 3). Napięcia potrzebne do zasilania nadajników linii wytwarzane są z niestabilizowanych napięć 9V AC. Z tego powodu w konstrukcji wykorzystano nadajnik 1488, który funkcjonuje poprawnie nawet przy napięciu zasilania niższym od znamionowego. Na rysunku 3 zaznaczono jedynie niezbędne połączenia. Używane są tu tylko trzy linie interfejsu szeregowego (na-



Rys. 2. Schemat ideowy przystawki video do C 64, umożliwiającej wyświetlanie 80 znaków w wierszu



Rys. 3. Interfejs szeregowy RS 232 C do COMMODORE C 64

dawanie, odbieranie i masa). Pokazane rozwiązanie może być wykorzystane w różnych zastosowaniach. W niektórych aplikacjach może się jednak okazać konieczne wykorzystanie pozostałych linii kontrolno-sterujących objętych standardem RS 232C. Commodore C64 może zasilać układy interfejsu RS 232C, jednak płytka video wymaga zasilania ze źródła zewnętrznego 5V/1A.

Oprogramowanie

Jak już wspomniano, Commodore C64 wymaga specjalnego programu obsługi płytki video. Żaden program napisany dla standardowej wersji mikrokomputera nie będzie sterować opisaną przystawką video bez wprowadzenia znacznych modyfikacji. Program sterujący należy napisać w języku ASSEMBLER. Wykorzystanie języka BASIC jest wprawdzie możliwe, lecz powoduje, że całe urządzenie działa wyjątkowo wolno.

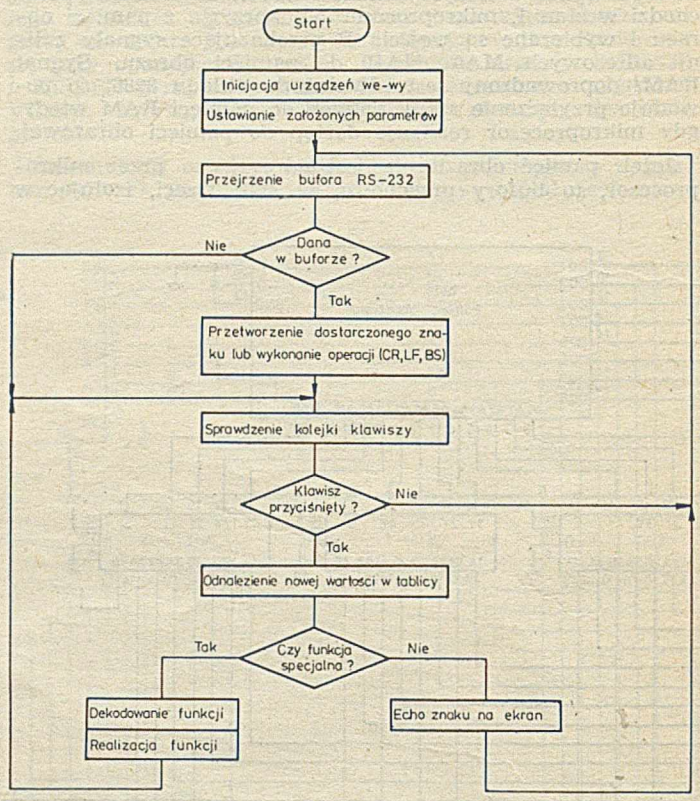
Program można umieścić w przewidzianej na płytce video pamięci EPROM (2716). Komputer będzie ją traktował jak typową kasetkę z oprogramowaniem (ang. cartrige ROM). Po włączeniu zasilania Commodore C64 szuka w zawartości ROM znaków CBM80. Jeśli je znajdzie (oznacza to, że przyłączona jest kasetka), przekazuje sterowanie do programu w ROM. Zgodnie z tą konwencją w początkowych kilku bajtach pamięci należy umieścić tekst CBM80 w kodzie ASCII stosowanym w rodzinie komputerów PET-COMMODORE.

Na rysunku 4 przedstawiono algorytm programu sterującego dla C64 jako terminala. Program ten najpierw inicjuje urządzenia we-wy. Trzeba pamiętać o zaprogramowaniu CRTC, uaktywnieniu sterowanej przerwaniami procedury przeglądania klawiatury, a także otwarciu kanałów klawiatury i portu szeregowego RS 232C. Należy również utworzyć tablicę konwersji kodów odczytywanych z klawiatury na znaki ASCII. Na koniec należy uaktywnić procedurę „auto-repeat", która powoduje samoczynne powtarzanie znaku przy trwającym nieco dłużej przyciśnięciu klawisza. Jeżeli terminal ma pracować w trybie dupleksowym, należy zablokować procedurę wyświetlającą na ekranie wprowadzony z klawiatury znak.

Jak wynika z pozostałej części przedstawionego algorytmu, program pobiera znaki z bufora odbiorczego kanału szeregowego RS 232C oraz wyświetla je na ekranie. W tym samym czasie odbiera on znaki z klawiatury, przekodowuje na ASCII i wysyła do bufora nadawczego kanału szeregowego. Jeśli terminal pracuje w trybie półdupleksowym (włączone jest lokalne echo), to dane z klawiatury są wysyłane także do pamięci obrazu.

W C64 przewidziano osiem specjalnych klawiszy funkcyjnych. Można je wykorzystać do realizacji funkcji związanych z obsługą terminala, np.:

f1: cofnięcie (ang. backspace)
f2: włączenie/wyłączenie lokalnego echa, czyli dupleks lub półdupleks
f3: przesłanie BREAK
f4: przejście na zapis wyłącznie dużych liter
f5: wymazanie zawartości ekranu
f6: powrót do zapisu normalnego (małe i duże litery)



Rys. 4. Algorytm blokowy programu obsługi terminala zbudowanego na bazie C 64

Uruchamianie konstrukcji

Przed włączeniem zasilania należy jeszcze raz sprawdzić poprawność wszystkich połączeń. We wstępnej fazie uruchamiania lepiej jest sprawdzić działanie układu bez wykorzystywania EPROM z programem sterującym. Komputer działa wtedy pod kontrolą standardowego systemu operacyjnego. Oznacza to, że do wyjścia video lub TV komputera Commodore C64 musi być przyłączony monitor lub odbiornik telewizyjny. Do pamięci komputera należy wprowadzić listę kodów inicjujących sterowanie CRT i przesyłać je kolejno do rejestrów od R0 do R15. Jeśli obwód działa prawidłowo, na przyłączonym do przystawki monitorze wyświetlają się 24 linie po 80 przypadkowych znaków. Znaki te można zmieniać przez wprowadzanie rozkazem POKE do pamięci obrazowej różnych wartości. Jeśli na ekranie pojawiły się znaki, ale wyświetlanie nie jest realizowane poprawnie, należy spróbować zmienić wartości w rejestrach CRTC. Jeśli monitor nie trzyma synchronizacji, należy sprawdzić rejestry R4 lub R0, ponieważ od umieszczonych w nich wartości zależą impulsy odchylania pionowego i poziomego. Jeżeli tak przeprowadzony test wypadnie pozytywnie, należy wyłączyć zasilanie oraz włożyć EPROM zawierający program obsługi.

ROLAND HOJCZAK
Ośrodek Badań Stanu Torów
Warszawa

Można by zapełnić rzekę deficytowego w naszym kraju papieru dyskutując nad rozlicznymi ograniczeniami, jakim musi stawić czoło użytkownik ZX SPECTRUM. W porównaniu z wprowadzanymi obecnie na rynek nowymi typami komputerów, dotkliwą bolączką dla stosunkowo szerokiej rzeszy użytkowników ZX Spectrum jest brak w nim wielu użytecznych instrukcji. INFORMATYKA publikowała już opis rozszerzonej normy języka BASIC — ANSI, w świetle której jest to niemal zupełnie inny (jakościowo) język.

Opisywany niżej program daleki jest od tak szeroko potraktowanych zmian. Tym niemniej, zestawiając obszar pamięci, który traci użytkownik, z dodatkowymi możliwościami, które mu daje, należy ocenić program BETA BASIC pozytywnie.

Program BETA BASIC

Dla większości mikrokomputerów oferowane są programy rozszerzające przewidziany przez producenta zestaw instrukcji języka BASIC. Dodatkowe instrukcje znacznie ułatwiają pracę na komputerze. Zalety takiego rozwiązania są szczególnie odczuwalne w przypadku współpracy z szybką pamięcią masową (np. dyskami elastycznymi). Wiąże się to z krótkim czasem wprowadzania programu rozszerzającego podstawowy zasób słów wykorzystywanego języka.

Dla ZX SPECTRUM takim programem jest program BETA BASIC firmy BETASOFT. W artykule opisana zostanie wersja 1.8 z 1984 r. Program ten ma ok. 9000 bajtów długości. Jest to istotne, gdyż musi on zostać wczytany do pamięci razem z programem użytkownika. Wczytanie programu użytkownika realizujemy instrukcją **MERGE**, po uprzednim wczytaniu programu BETA BASIC. Taki sposób wczytania zapewnia automatyczny start naszego programu.

Istnieją dwa sposoby wywoływania dodatkowych instrukcji. Pierwszy polega na wykorzystaniu znaków graficznych; drugi realizowany jest przez instrukcję **FN**, pełniącą w tym przypadku rolę pośrednika. Aby nie pozbawiać użytkownika możliwości korzystania ze standardowych znaków graficznych, wprowadzono instrukcję **KEYWORDS**. I tak, instrukcja **KEYWORDS 1** przyporządkowuje klawiszom nowe słowa, a **KEYWORDS 0** standardowe znaki graficzne (wszystko z kursorem G).

Łącznie poprzez znaki graficzne można odwołać się do 35 dodatkowych instrukcji:

A-**ALTER**, C-**CLOCK**, D-**DO**, E-**ELSE**, F-**FILL**, G-**GET**, I-**EXIT IF**, J-**WHILE**, K-**UNTIL**, L-**LOOP**, M-**SORT**, N-**ON ERROR**, O-**ON**, P-**DPOKE**, R-**ROLL**, S-**SCROLL**, T-**TRACE**, U-**USING**, V-**RND**, W- **INKEY$**, Y-**PI**, Y-**FN**, Z-**POINT**, 0-**POP**, 1-**DEF PROC**, 2-**PROC**, 3-**END PROC**, 4-**RENUM**, 6-**AUTO**, 7- **DELETE**, 8- **KEYWORDS**, !-**DEF KEY**, $-**KEY IN**, %-**EDIT** (dla trybu graficznego), &-**JOIN**, 0-**EDIT** (dla trybu niegraficznego).

Przez FN wywołujemy 21 kolejnych, dodatkowych instrukcji: a-**AND**, c-**COSE**, f-**FILLED**, i-**INSTRING**, m-**MEM**, n-**NUMBER**, o-**OR**, p-**DPEEK**, r-**RNDM**, s-**SINE**, v-**MOD**, x-**XOR**, b$-**BIN$**, c$-**CHAR$**, h$-**HEX$**, k$-**SCRN$**, m$-**MEMOR$**, s$-**STRING$**, t$-**TIME$**, u$-**USING$**. Dużym ułatwieniem jest fakt, że zawsze możemy odwołać się (przez ENTER) do strony, na której podany jest opis argumentów instrukcji wywoływanych przez FN. Dla przykładu, zapis **FN m$ / /** oznacza kolejność wprowadzania z klawiatury, a zarazem informuje, że instrukcja jest bezargumentowa. Nazwa instrukcji, w tym przypadku **MEMOR$**, pojawi się na ekranie po wciśnięciu klawisza „(" (otwarcie nawiasu).

Ogólnie biorąc, nowe instrukcje można pogrupować następująco:

- poprawiające komfort pracy z komputerem
- upraszczające strukturę programu
- pozwalające lepiej wykorzystać pamięć komputera
- stwarzające nowe możliwości graficzne.

Jedna z ciekawszych instrukcji jest **CLOCK**. Ma ona 8 trybów, dających następujące kombinacje: wskazywanie czasu, skok do zdefiniowanego przez użytkownika podprogramu obsługi i włączenie alarmu dźwiękowego. Tryb 7 zapewnia jednoczesną realizację wszystkich trzech możliwości. Normalnie zegar startuje z czasem zerowym, licząc od chwili włączenia zasilania. Odmiana tej instrukcji jest **TIME$**. Po wywołaniu tej instrukcji wskazywany jest czas momentu wywołania.

AUTO ⟨start⟩/⟨start, krok⟩ uwalnia użytkownika od kłopotliwego wprowadzania kolejnego numeru linii programu. Jeżeli nie podamy parametru krok, to standardowo jest on ustawiony na 10.

DELETE (nr wiersza) TO (nr wiersza) usuwa wybrany fragment programu.

Instrukcja warunkowa „jeśli" ma postać **IF** ⟨wyrażenie logiczne⟩ **THEN** ⟨instrukcja⟩ /:**ELSE** ⟨instrukcja⟩⟩. Cecha charakterystyczna jest konieczność umieszczenia dwukropka przed ELSE.

W BETA BASIC wprowadzono na wzór języka PASCAL, instrukcje iteracyjne tworzone za pomocą słów: **DO, WHILE, UNTIL, LOOP, EXIT IF**. Również z PASCALA zaczerpnięte są trzy instrukcje do tworzenia i wywoływania procedur: **DEF PROC, PROC, END PROC**. Konstrukcja wyliczanego skoku oraz wyliczanego wywołania podprogramu jest odmienna od typowej i ma postać: **GOTO ON, GOSUB ON**.

ON ERROR (nr linii) pozwala na obsługę błędu, przez skok do podprogramu zaczynającego się od podanego jako parametr numeru linii.

POP (zmienna) usuwa ze stosu adres. Dzięki temu można wychodzić z procedur i pętli, nie robiąc zamieszania w stosie. Metodę taką stosuje się w podprogramach do powrotu do programu głównego bez używania RETURN. Czasami jest to bardzo wygodne.

SORT porządkuje tablicę numeryczną lub łańcuch od wartości najmniejszej do największej (ewentualnie wykorzystując kody ASCII).

SORT INVERSE sortuje odwrotnie. Przykładowa konstrukcja: **SORT a$ (1 TO 7)**.

CHAR$ (liczba) zamienia liczbę z przedziału 1—2^{16} na ciąg o długości d w ó c h liter. Pozwala to na zmniejszenie obszaru pamięci zajmowanego przez tablicę numeryczną. **NUMBER** (dwie litery) działa w kierunku odwrotnym.

DEC zamienia liczby z postaci szesnastkowej na dziesiętną.

HEX$ działa w przeciwnym kierunku.

DPEEK wprowadza zawartość dwóch kolejnych komórek pamięci. Przyjęta jest tu konwencja, że pod podanym jako parametr adresem umieszczony jest mniej znaczący bajt, a pod adresem o 1 większym — bardziej znaczący. Oba bajty łączone są w liczbę 16-bitową.

DPOKE ustawia zawartość dwóch kolejnych komórek pamięci według podanej jako parametr liczby 16-bitowej (0 ... 65535).

MEM / / podaje jako wynik liczbę wolnych bajtów pamięci. Jest odpowiednikiem sekwencji PRINT 65535 — — USR 7962.

INSTRING (start, a$, b$) szuka w łańcuchu a$ łańcucha b$ zaczynając od elementu łańcucha a$ start.

**STRING$** (n, a$) powtarza łańcuch a$ zadany przez parametr n liczbę razy.

USING jest instrukcją formatującą wyprowadzaną informację. Przykładowa konstrukcja PRINT USING ("## · ·##" : wyrażenie).

ROLL kierunek :⟨x, y : szer, wys⟩ umożliwia przesuwanie wycinka ekranu (okna) w podstawowych kierunkach o jeden punkt. Kierunek określany jest jako liczba 5 ... 8 według zamieszczonych na klawiaturze strzałek: x i y jak w PLOT; szer = 1 ... 32, wys = 1 ... 176.

SCROLL przesuwa obraz na ekranie o wiersz do góry. SCROLL kierunek — przemieszcza obraz na ekranie o jeden punkt w zadanym kierunku. To, co wychodzi poza ekran, jest kasowane.

Oprócz wprowadzenia dodatkowych instrukcji w BETA BASIC rozszerzono również możliwości niektórych instrukcji dotychczasowych.

PLOT x, y ;⟨łańcuch⟩ wpisuje w dowolnym miejscu ekranu, z dokładnością do jednego punktu, łańcuch o długości do 32 znaków. Warunkiem koniecznym dla funkcjonowania instrukcji jest wciśnięcie **PLOT** przy pracy w trybie niegraficznym (KEYWORDS 0). Możliwości PLOT i DRAW dodatkowo poszerzają zmienne skalowania **XOS**, **YOS**, **XRG**, **YRG**. Wartości początkowe wynoszą odpowiednio 0, 0, 256, 176. Wartości tych zmiennych nie są zmieniane przez RUN i CLEAR. XOS i YOS powodują przesunięcie początku układu współrzędnych, natomiast XRG i YRG — przesunięcie wzdłuż osi x i y.

Nawet tak pobieżny przegląd możliwości programu BETA BASIC powinien przekonać Czytelników o jego dużej użyteczności. Przy tworzeniu nieco dłuższych programów w pełni rekompensuje się 9 KB pamięci zajętych przez BETA BASIC. Wynika to z możliwości zastąpienia wielu instrukcji języka ZX SPECTRUM BASIC czasem nawet pojedynczą instrukcją BETA BASIC.

JERZY KRYSTEK
ADAM MIELCZAREK
Zabrze

Tematyka grafiki mikrokomputerowej przyciąga coraz więcej uwagi. Planowaliśmy poświęcić jej więcej uwagi niż nam się w 1985 r. udało. Dlatego też z pewnością poświęcimy jej jeszcze sporo miejsca w przyszłości, choć zapewne formuła wysokonakładowego pisma, jakim ma się stać mikroKLAN spowoduje, że będziemy musieli opracować bardziej przystępną dla szerszego kręgu czytelników formę. Będziemy jednak zapewne mogli, pisząc o grafice, również ją pokazać.

Antoni Urban w kolejnym materiale związanym z normą jądra graficznego GKS pisze o rysowaniu różnych łuków — w tym okręgów i paraboli. Jest to zagadnienie, z którym musimy sobie jakoś poradzić niemal przy każdej grafice. Efektywny algorytm (tu akurat liczy się szybkość działania i dokładność) to główny kierunek poszukiwań programistów wykorzystujących mikrokomputery.

Łuki krzywych gładkich

Rysowanie łuków krzywych gładkich nie jest objęte normą GKS. Wśród prymitywów nie ma reprezentacji graficznej łuków okręgu, elipsy, hiperboli, paraboli, nie mówiąc już o bardziej skomplikowanych krzywych. Norma GKS wymienia jedynie prymityw o nazwie Generalized Drawing Primitive (w skrócie G.D.P.), pod którym autorzy implementacji mogą przewidzieć krzywe, które ich zdaniem będą wykorzystywane przez użytkowników jądra graficznego. W przypadku realizowania implementacji normy GKS dla sprzętu mikrokomputerowego ważny jest jeszcze jeden czynnik: możliwości obliczeniowe sprzętu. Czynnik ten nie tylko decyduje o wyborze takich a nie innych krzywych, lecz również zmusza do szukania najbardziej optymalnych algorytmów interpolacji. I to właśnie, skromnym zdaniem autora, jest bakcylem mikroinformatyki, który raz połknięty zaraża na całe życie.

Rozpatrzmy następujący przykład: jak narysować okrąg, gdy brak jest interpolatora kołowego? Problem rozwiązać można na wiele sposobów.

1. Okrąg o środku **S(a, b)** i promieniu **r** zastępowany jest wielokątem o **n** wierzchołkach, których współrzędne wyrażają się wzorami:

$$x_i = a + r \cdot \cos(i \cdot \varphi)$$
$$y_i = b + r \cdot \sin(i \cdot \varphi)$$

gdzie **i** przyjmuje kolejne wartości od **0** do **n—1**. W opisie nie uwzględniono kątów początkowego i końcowego łuku.

2. Idea rozwiązania jest identyczna jak w poprzednim rozwiązaniu, przy czym współrzędne kolejnych wierzchołków wielokąta obliczane są w sposób rekurencyjny według wzorów:

$$x' = (x-a)\cdot\cos\varphi - (y-b)\cdot\sin\varphi + a$$
$$y' = (y-b)\cdot\cos\varphi + (x-a)\cdot\sin\varphi + b$$

Ponieważ w tym przypadku kąt φ jest dostatecznie mały, można zamiast funkcji trygonometrycznych zastosować przybliżenie wynikające z rozwinięcia szeregu Fouriera:

$$\sin\varphi = \varphi - \frac{\varphi^3}{6} + \dots$$

$$\cos\varphi = 1 - \frac{\varphi^2}{2} + \dots$$

3. Kreślone są jednostkowe odcinki między sąsiednimi punktami w takim kierunku, by zachować minimalną odległość od środka okręgu. Rozwiązanie to jest podobne do rozwiązania następnego, ale wzór, według którego wybierany jest optymalny kierunek, nie gwarantuje zadowalającej szybkości.

$$d^2 = (x-a)^2 + (y-b)^2$$

Żadne z dotąd omówionych rozwiązań nie daje zadowalających rezultatów; w pierwszym konieczne jest wyliczanie wartości funkcji sinus i cosinus, wszystkie odwołują się do mnożenia.

Okazuje się, że można narysować okrąg wykonując tylko dodawanie, odejmowanie i porównywanie liczb całkowitych. Czytelnik łatwo stwierdzi, że jest to rozwiązanie najlepsze, zapewniające największą dokładność interpolacji, dużą szybkość rysowania, a co najważniejsze — nie powodujące występowania błędów przy źle dobranych parametrach wejściowych. Procedura wymaga zadeklarowania współrzędnych trzech punktów na płaszczyźnie (środek okręgu S, początek łuku A, koniec łuku B) oraz kierunku poruszania się po okręgu — zgodnie z ruchem wskazówek zegara (k = —1) lub przeciwnie do ruchu wskazówek zegara (k = +1). Ze zrozumiałych względów końce łuku nie mogą być równe środkowi okręgu. Może być natomiast tak, że początek i koniec łuku są tym samym punktem — rysowany jest wtedy pełny okrąg. Algorytm interpolacji kołowej w niezawodny sposób bada warunek zakończenia rysowania, nawet wtedy, gdy punkt końcowy zadeklarowany został w innej odległości od środka niż początek łuku. Interpolacja kończy się w momencie przecięcia półprostej $SB^{\rightarrow}$. Algorytm wykorzystuje ten sam aparat matematyczny, który wymieniono przy demonstrowaniu interpolatora liniowego (zob. „Wielolinia", mikroKLAN 10'85).

Aby lepiej zrozumieć ten algorytm, należy pamiętać, że funkcja

$$(x, y) = (x-a)^2 + (y-b)^2 - r^2$$

przedstawia okrąg, a funkcja:

$$g(x, y) = \begin{vmatrix} x-a & y-b \\ x_1-a & y_1-b \end{vmatrix} \cdot k$$

przedstawia prostą łączącą środek okręgu i koniec łuku. W tym ostatnim wzorze uwzględniono kierunek interpolacji z tego względu, by zmiana znaku wartości przyjmowanej przez funkcje g z „+" na „—" oznaczała przecięcie półprostej (a nie prostej) $SB^{\rightarrow}$. Na rysunku 1 przedstawiono graf algorytmu.

Po zapoznaniu się z procedurą interpolacji kołowej należy powrócić do języka normy GKS, by zobaczyć składnię instrukcji rysowania łuków okręgu.

GDP (n, p)

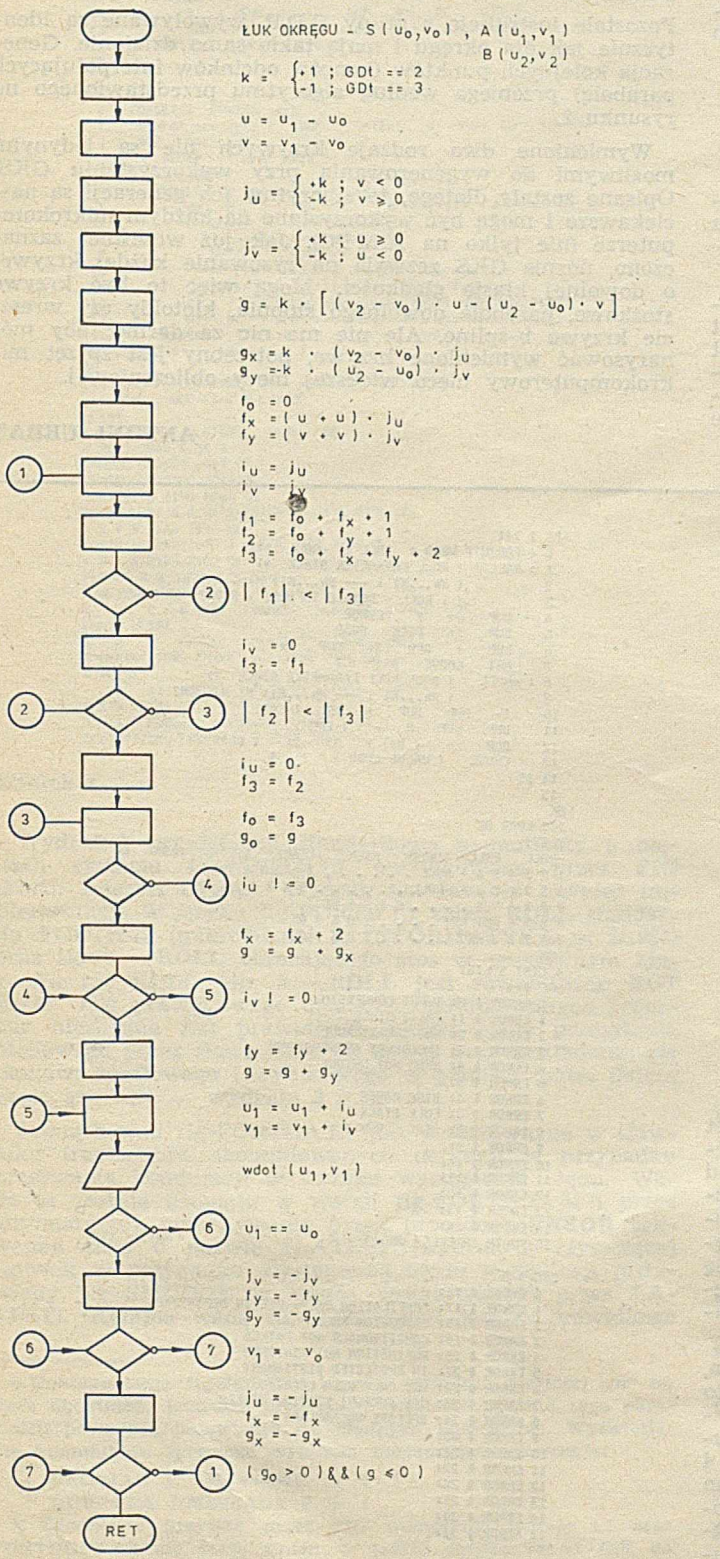
gdzie: **int n** — liczba punktów, **float p [ ] [2]** — współrzędne punktów **A, S, B**

Pojedynczą instrukcją GDP można narysować kilka łuków okręgu pod warunkiem, że wszystkie rysowane są w tym samym kierunku.

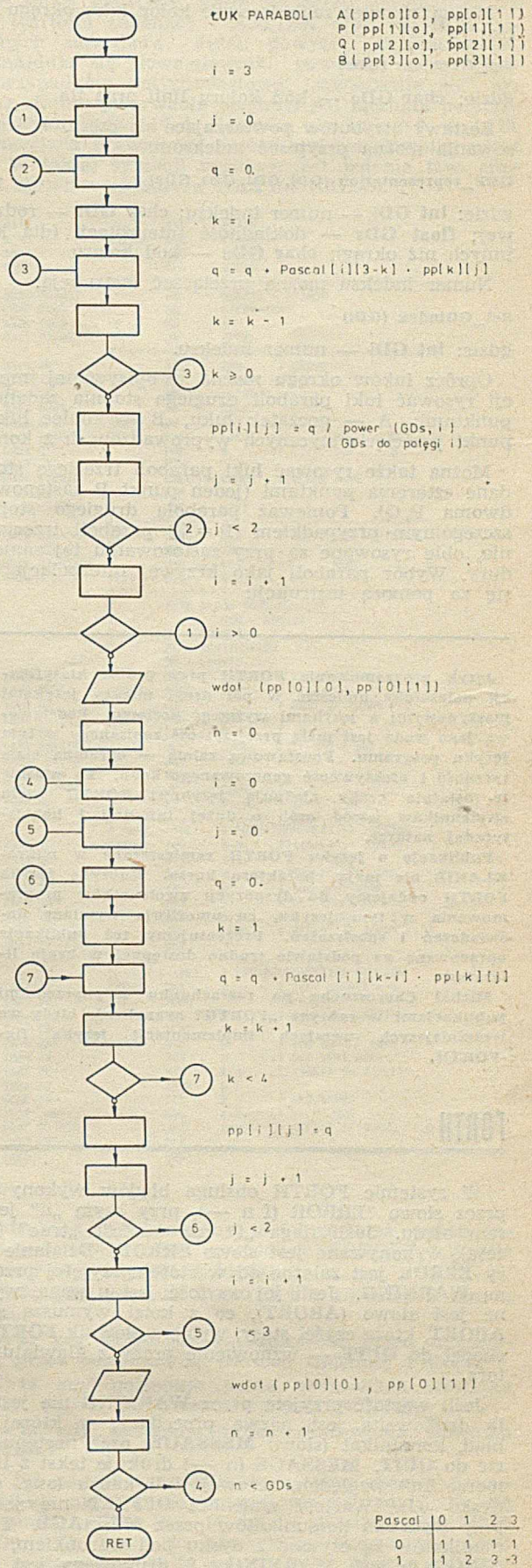
Kierunek interpolacji zadaje się instrukcją:

Set_GDtype (GDt)

gdzie: **char GDt** — rodzaj krzywej, **GDt = 2** — łuk okręgu rysowany zgodnie z ruchem wskazówek zegara (CW), **GDT = 3** — łuk okręgu rysowany przeciwnie do ruchu wskazówek zegara (CCW).



Rys. 1



Pascal	0	1	2	3
0	1	1	1	1
1	1	2	3	-3
2	1	3	-6	3
3	1	-3	3	-1

Rys. 2

Można również zadeklarować kolor łuku okręgu instrukcją:

Set_GDcolour (GDc)

gdzie: **char GDc** — kod koloru linii oraz tła

Zestawy atrybutów powtarzające się często w trakcie rysowania można przypisać indeksom:

GDP_representation (GDi, GDt, GDs, GDc)

gdzie: **int GDi** — numer indeksu; **char GDt** — rodzaj krzywej; **float GDs** — dokładność interpolacji (dla krzywych innych niż okrąg); **char GDc** — kod koloru

Numer indeksu można przełączać instrukcją:

Set_GDindex (GDi)

gdzie: **int GDi** — numer indeksu.

Oprócz łuków okręgu można w opisywanej implementacji rysować łuki paraboli drugiego stopnia zadane trzema punktami: A — początek łuku, B — koniec łuku, P — punkt przecięcia stycznych wyprowadzonych z końców.

Można także rysować łuki paraboli trzeciego stopnia zadane czterema punktami (jeden punkt P zastępowany jest dwoma P, Q). Ponieważ parabola drugiego stopnia jest szczególnym przypadkiem (P = Q) paraboli trzeciego stopnia, obie rysowane są przy zastosowaniu tej samej procedury. Wybór paraboli jako krzywej interpolacji dokonuje się za pomocą instrukcji:

Set_GDtype (GDt)

gdzie: **char GDt** — rodzaj krzywej, **GDt = 0** — parabola rysowana punktami, **GDt = 1** — parabola rysowana odcinkami.

Atrybut GDsize oznacza w tym przypadku ilość punktów przybliżających parabolę, a więc dokładność interpolacji.

Set_GDsize (GDs)

gdzie: **float GDs** — dokładność interpolacji.

Pozostałe instrukcje z grupy G.D.P. wywoływane są identycznie jak dla okręgu i mają takie samo działanie. Generacja kolejnych punktów (końców odcinków interpolujących parabolę) przebiega według algorytmu przedstawionego na rysunku 2.

Wymienione dwa rodzaje krzywych nie są jedynymi możliwymi do wygenerowania przy wykorzystaniu GKS. Opisane zostały dlatego, że algorytmy ich generacji są najciekawsze i mogą być wykorzystane na każdym mikrokomputerze (nie tylko na IBM PC). Jak już wcześniej zaznaczono, norma GKS zezwala na rysowanie każdej krzywej o dowolnej klasie gładkości. Mogą więc to być krzywe stożkowe, parabole dowolnego stopnia, klotoidy czy wreszcie krzywe b-spline. Ale nie ma nic za darmo; aby móc narysować wymienione krzywe, potrzebny jest sprzęt mikrokomputerowy nieco większej mocy obliczeniowej.

ANTONI URBAN

Język programowania FORTH przy próbie klasyfikacji należałoby umieścić w pół drogi między językami maszynowymi a językami wyższego poziomu. Podstawową jego wadą jest mała przejrzystość zapisanego w tym języku programu. Podstawową zaletą — ogromna elastyczność i efektywność generowanego kodu. To właśnie te ostatnie cechy zjednują językowi FORTH wielu zwolenników wśród osób o dużej inwencji i hobbistycznej naturze.

Publikacje o języku FORTH zamieszczane w mikroKLANIE nie mają charakteru kursu. Rubrykę języka FORTH oddajemy do dyspozycji zwolenników programowania w tym języku, co umożliwia wymianę doświadczeń i spostrzeżeń. Prezentujemy też publikacje opracowane na podstawie trudno dostępnej w kraju literatury.

Michał Choroszucha po rozrachunku z poprzednimi publikacjami w rubryce „FORTH" sygnalizuje błędy we wcześniejszych wersjach implementacji języka fig-FORTH.

FORTH

W systemie FORTH obsługa błędów wykonywana jest przez słowo ?ERROR (f n ---), przy czym „n" jest numerem błędu. Jeśli flaga „f" ma wartość „true" (różną od zera), wykonywane jest słowo **ERROR**. Działanie procedury **ERROR** jest zależne od wartości przyjętej przez zmienną **WARNING**. Jeśli jej wartość jest ujemna, wykonywane jest słowo (**ABORT**), co z kolei wymusza sekwencję **ABORT**, która czyści stosy, wybiera słownik **FORTH** i przechodzi do **QUIT** — wznowienie pracy z klawiatury operatora.

Jeśli wartość przyjęta przez **WARNING** nie jest ujemna, to drukowana jest nazwa procedury, w której wykryto błąd, komunikat (słowo **MESSAGE**) oraz następuje przejście do **QUIT**. **MESSAGE** (n ---) drukuje tekst z linii o numerze „n" względem zerowej linii kadru (ang. screen) 4 dysku 0 [1]). Wartość zmiennej **OFFSET** nie jest istotna przy wydruku komunikatów przez **MESSAGE**. Teksty komunikatów są czytane z dysku pod warunkiem, że **WARNING** > 0; jeśli **WARNING** = 0, drukowany jest tylko numer komunikatu.

```
SRC # 241
  0 ( FORTH79 WORD :  ROLL ; -ROLL  **)
  1 : ROLL    ( ROLL PARAMETER STACK  *)
  2           ( Nk...N1 i --- Nk...Ni+ Ni-...N1 Ni )
  3           ( i ROLL = ROTATE TOPMOST i DIGITS ON STACK )
  4     DUP   0=   9   ?ERROR
  5     DUP   1+   PICK   SWAP
  6     DUP   +   SP@   2+   DUP   2+
  7     ROT   CMOVE   DROP   ;
  8 : -ROLL     ( BACK ROLL PARAMETER STACK  *)
  9           ( Nk...N1 i --- Nk...Ni+ N1 Ni...N2 )
 10     1-   DUP   DUP   +   )R   (HEAP)
 11     DUP   DUP   R   -   ( SRC )
 12     DUP   2-   ( DST )   R)   2+   ( QUAN )
 13     CMOVE   ( VALUE ADDR )   !   ;
 14 ;S
 15
OK
240 LOAD OK
VLIST
-ROLL   ROLL   !HEAP   PICK   (HEAP)   DEPTH   TASK   OK
1 2 3 4 5 6 4 ROLL . . . . . 3 6 5 4 2 1 OK
1 2 3 4 5 6 4 -ROLL . . . . . 5 4 3 6 2 1 OK
9 8 7 6 ROT . . . . 8 6 7 9 OK
9 8 7 6 ROT ROT . . . . 7 8 6 9 OK
9 8 7 6 3 -ROLL . . . . 7 8 6 9 OK
 OK
4 LIST 5 LIST
SRC # 4
  0 ERROR # 0: INPUT CONVERSION
  1 ERROR # 1: EMPTY STACK
  2 ERROR # 2: DICTIONARY FULL
  3 ERROR # 3: INCORRECT ADDRESS MODE
  4 ERROR # 4: NAME NOT UNIQUE
  5 ERROR # 5:
  6 ERROR # 6: DISC RANGE ?
  7 ERROR # 7: FULL STACK
  8 ERROR # 8: DISC ERROR !
  9 ERROR # 9:
 10 ERROR # 10:
 11 ERROR # 11:
 12 ERROR # 12:
 13 ERROR # 13:
 14 ERROR # 14:
 15 FORTH INTEREST GROUP                          MAY 1, 1979

SRC # 5
  0 ERROR # 16:
  1 ERROR # 17: COMPILATION ONLY, USE IN DEFINITION
  2 ERROR # 18: EXECUTION ONLY
  3 ERROR # 19: CONDITIONALS NOT PAIRED
  4 ERROR # 20: DEFINITION NOT FINISHED
  5 ERROR # 21: IN PROTECTED DICTIONARY
  6 ERROR # 22: USE ONLY WHEN LOADING
  7 ERROR # 23: OFF CURRENT EDITING SCREEN
  8 ERROR # 24: DECLARE VOCABULARY
  9 ERROR # 25:
 10 ERROR # 26:
 11 ERROR # 27:
 12 ERROR # 28:
 13 ERROR # 29:
 14 ERROR # 30:
 15 ERROR # 31:
OK
```

Wydruk 1

```
CR 240 LIST 241 LIST

SRC # 240
  0 ( ?LATEST   FORGET  **)
  1 : ?LATEST   ( ---   Verify Dictionary Pointer  *)
  2    LATEST   C@   [ HEX ] 20   AND
  3       IF   ( Bad entry - Compilation error )
  4          LATEST   DUP   DP   !
  5          PFA   LFA   @   CURRENT   @   !
  6       ENDIF   ;
  7 : FORGET   ( ---   FORGET name  *)
  8    CURRENT   @   CONTEXT   @   -   [ HEX ]   18   ?ERROR
  9    [COMPILE]   '   DUP   FENCE   @   U<   15   ?ERROR
 10    ( PFA'   Following modified )   NFA   VOC-LINK
 11       BEGIN
 12          @   ( NFA' pvla )   DUP   0=   0E   ?ERROR
 13          2DUP   U<   ( NFA' ( pvla ? )   0=
 14       UNTIL   ( NFA' > pvla )
 15 --)

SRC # 241
  0 ( FORGET -   CONT'D  **)
  1    ( NFA' new_pvla )   DUP   CONTEXT   @   U<   0D   ?ERROR
  2    DUP   VOC-LINK   !
  3       BEGIN   ( NFA' pvla )
  4          -DUP
  5       WHILE   ( Vocabulary found )
  6          )R   R   2-   )R   R   ( NFA' pla )   @
  7             BEGIN   ( NFA' nfa/pnfa )
  8                2DUP   SWAP   U<   0=
  9             WHILE   ( NFA' (= nfa )
 10                PFA   LFA   @   ( NFA' new_nfa )
 11             REPEAT   ( Until  nfa ( NFA'  found )
 12          R)   !   ( Store new vocab. limit NFA into PLA )
 13          R)   @   ( NFA' new_pvla   Process next vocabulary )
 14       REPEAT   ( Until end of vocab. link chain )
 15    ( NFA' )   DP   !   ;   ;S
OK
240 LOAD FORGET ERROR # 4: NAME NOT UNIQUE  OK
: & HERE . VOC-LINK @ . ; OK
VLIST
&   FORGET   ?LATEST   TASK   OK
& 2190 2064 : GG ; OK
SMUDGE & 2199 2064 OK
?LATEST & 2190 2064 OK
?LATEST & 2190 2064 OK
VOCABULARY A A DEFINITIONS & 219E 219C OK
: X ; & 21A6 219C OK
FORTH DEFINITIONS : C ; & 21AE 219C OK
: D ; & 21B6 219C OK
FORGET D & 21AE 219C OK
A VLIST
X      C   A   &   FORGET   ?LATEST   TASK   OK
FORTH VLIST
C   A   &   FORGET   ?LATEST   TASK   OK
A DEFINITIONS FORGET X & 219E 219C OK
A VLIST
    A   &   FORGET   ?LATEST   TASK   OK
FORTH VLIST
A   &   FORGET   ?LATEST   TASK   OK
DEFINITIONS FORGET A & 2190 2064 OK
BYE
```

Wydruk 2

Wydruk 1 przedstawia standardowe komunikaty o błędach systemu fig-FORTH 1.1 opublikowane przez FIG (Forth Interest Group). Wydruk ten zawiera również implementację w języku fig-FORTH 1.1 słowa **ROLL** standardu FORTH 79 (mikroKLAN 2, INFORMATYKA nr 2/1984) oraz słowa **—ROLL** obracającego stos w przeciwnym kierunku niż ROLL (np. 3 **—ROLL** jest równoważne **ROT ROT**). Dla prawidłowego działania opublikowanych procedur niezbędne jest prawidłowe wykonywanie przesyłania blokowego przez słowo **CMOVE** nawet przy nakładaniu się bloków: źródłowego i wynikowego w pamięci (patrz dalsza część artykułu — wydruki 3 i 4).

Jedną z wad fig-FORTHA 1.1 jest pozostawienie w słowniku fragmentów skompilowanych definicji w przypadku przerwania kompilacji w wyniku wystąpienia błędu. Wada ta została usunięta w wersji fig-FORTH 2.1 M [2]) przez automatyczne wykonywanie przez procedurę **ERROR** sekwencji słów o nazwie **?LATEST** (wydruk 2) czyszczącej słownik w przypadku wystąpienia błędu kompilacji (ustawiony bit SMUDGE w słowie wskazywanym przez **LATEST**). Kolejną wadą fig-FORTHA 1.1 jest nieprawidłowe działanie słowa **FORGET**, które w wersji oryginalnej nie zmienia wartości zmiennej VOC-LINK oraz pól PLA poszczególnych słowników. Jeżeli powyżej „zapomnianego" słowa znajdują się słowa-słowniki, powoduje to w większości przypadków natychmiastowe „zawieszenie się" systemu na skutek przerwania ciągłości pól łącznika poszczególnych definicji (od PLA słownika do link = 0 w LIT), co uniemożliwia prawidłowe przeszukiwanie słownika przez (FIND). W takiej sytuacji możliwy jest jedynie tzw. zimny start systemu od adresu HEX 100 (COLD).

```
0145 =          NEXT    EQU     0145H
0303 =          QTERM   EQU     0303H
16AA =          PCR     EQU     16AAH
030B            ORG     030BH
                ;---------------------------------------
030B 8243D2             DB      82H,'C','R'+80H
030E F702               DW      QTERM-0CH
0310 1203       CR      DW      $+2
0312 C32B03             JMP     XPCR
                ;---------------------------------------
0315 85                 DB      85H     ; CMOVE
0316 434D4F56           DB      'CMOV'
031A C5                 DB      'E'+80H
031B 0B03               DW      CR-5
031D 1F03       CMOVE   DW      $+2
031F 6960               MOV     L,C     !       MOV     H,B
0321 C1D1               POP     B       !       POP     D
0323 E3                 XTHL
0324 CD881A             CALL    CMOVX   ; BLOCK MOVE
0327 C1                 POP     B
0328 C34501             JMP     NEXT
                ;---------------------------------------
032B 210000     XPCR    LXI     H,0     ; CLEAR OUT VARIABLE
032E 22BA3B             SHLD    3BBAH
0331 C3AA1600           JMP     PCR     !       NOP
0335 =          XEND    EQU     $
                ;---------------------------------------
                ;       BYTE BLOCK MOVE ROUTINE
                ;               MCH - 850222
                ;       HL = SOURCE
                ;       DE = DESTINATION
                ;       BC = LENGTH
1A88                    ORG     1A88H   ; 1ST FREE CELL
1A88 78B1       CMOVX   MOV     A,B     !       ORA     C
1A8A C8                 RZ              ; QUIT IF #CHRS = 0
1A8B 7C         CMOV1   MOV     A,H
1A8C BA                 CMP     D       ; (H) - (D)
1A8D DA941A             JC      CMOV3   ; IF SRC < DST
1A90 C2991A             JNZ     CMOV2   ; IF >
1A93 7DBB               MOV     A,L     !       CMP     E
1A95 DA941A             JC      CMOV3   ; IF <
1A98 C8                 RZ              ; IF =
                ;---------------------------------------
                ; (HL) > (DE)   ; POST-INCREMENTING VERSION
1A99 7E         CMOV2   MOV     A,M     ; ((DE)) <- ((HL))
1A9A 12                 STAX    D
1A9B 23                 INX     H       ; INC SOURCE ADDR
1A9C 13                 INX     D       ; INC DEST ADDR
1A9D 0B                 DCX     B       ; DEC #CHRS
1A9E 78B1               MOV     A,B     !       ORA     C
1AA0 C2991A             JNZ     CMOV2   ; REPEAT IF #CHRS <> 0
1AA3 C9                 RET
                ;---------------------------------------
                ; (HL) < (DE)   ; PRE-DECREMENTING VERSION
1AA4 09         CMOV3   DAD     B       ; LAST SOURCE + 1
1AA5 EB                 XCHG
1AA6 09                 DAD     B       ; LAST DESTINATION + 1
1AA7 EB                 XCHG
1AA8 2B         CMOV4   DCX     H       ; DEC SOURCE ADDR
1AA9 1B                 DCX     D       ; DEC DEST ADDR
1AAA 7E                 MOV     A,M     ; ((DE)) <- ((HL))
1AAB 12                 STAX    D
1AAC 0B                 DCX     B       ; DEC #CHRS
1AAD 78B1               MOV     A,B     !       ORA     C
1AAF C2A81A             JNZ     CMOV4   ; REPEAT IF #CHRS <> 0
1AB2 C9                 RET
1AB3                    END
```

Wydruk 3

Na wydruku 2 przedstawiony został również sposób zainstalowania nowej wersji procedury **FORGET.** Wersja 2.1 M wykrywa następujące błędy:

ERROR # 24 gdy słowniki **CONTEXT** i **CURRENT** nie są równe,

ERROR # 21 gdy słowo jest zabezpieczone przed zniszczeniem przez odpowiednią wartość zmiennej **FENCE**,

ERROR # 14 gdy następuje próba skasowania słownika FORTH (nawet po zmianie wartości FENCE),

ERROR # 13 gdy następuje próba skasowania słownika **CONTEXT**.

Jak to już zostało zasygnalizowane, standardowa procedura **CMOVE** nie kopiuje prawidłowo nakładających się bloków pamięci. Nie jest to błąd; standard FORTH 79 przewiduje dwa słowa do przesyłania blokowego: słowo **CMOVE** przesyłające bajty od początku bloku (od najmniejszego adresu) oraz słowo **<CMOVE**, które przesyła

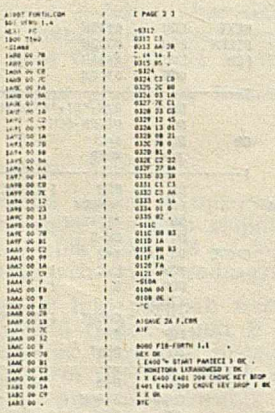
---

[1]) Dokumentacja fig-FORTHA 1.1 nie ogranicza wartości „n" do liczb dodatnich, jednakże nieparzyste wartości ujemne tego parametru powodują pojawianie się błędnych wydruków. Wyświetlanie komunikatu zerowego wymusza następująca sekwencja:

[DECIMAL 2 ' MESSAGE 12 + !
' DUP ' CFA ' MESSAGE 8 + !

[2]) Zgodnie z przyjętą przez FIG numeracją wersje 1.1 oraz FORTHDOS 1.2 nie mają zmian w jądrze języka — od LIT do TASK — przypis autora

bajty od największego adresu w kierunku adresów malejących (od końca bloku). W wersji 2.1 M przyjęta została zasada automatycznego określenia kierunku przesyłania blokowego. Analogiczne rozwiązanie zostało przedstawione na wydruku 3 w wersji INTEL 8080, przy czym ze względu na brak miejsca w pamięci w jądrze fig-FORTHA 1.1 część procedury została dopisana po słowie TASK. Przykładową sesję tworzenia nowej wersji języka przedstawia wydruk 4.



Wydruk 4

W odcinku sprzed dwóch miesięcy zostały opublikowane m. in. procedury VLIST, ?SPACES i TAB. Ponieważ nie zostało to wystarczająco podkreślone, pragnę wyjaśnić, iż żadna z tych procedur nie będzie działać prawidłowo, jeśli słowo CR nie będzie zerować zmiennej OUT. Odpowiednie zmiany podaje wydruk 3, a sposób ich generowania — wydruk 4.

Jak widać z wydruku 4, przy tworzeniu nowych wersji i języka z pliku CP/M typu .COM obowiązują następujące zasady:

— zmienia się wartości początkowe dla zmiennych FENCE (HEX 110), DP (11E) i VOC-LINK (120), przy czym dla DP przyjmuje się ostatnią wartość HERE (koniec słownika, z definicji : HERE DP @ ;),

```
                ;       SUBROUTINE USED BY - AND <
                ;                       ; (HL) <- (HL) - (DE)
083C                    ORG     083CH
083C 7D         SSUB    MOV     A,L     ; LB
083D 93                 SUB     E
083E 6F                 MOV     L,A
083F 7C                 MOV     A,H     ; HB
0840 9A                 SBB     D
0841 67                 MOV     H,A
0842 C9                 RET
                ;-------------------------------------------
0144 =          HPUSH   EQU     0144H
0861 =          LESS    EQU     0861H
087E                    ORG     087EH
087E 82                 DB      82H     ; U< ( UNSIGNED < )
087F 55                 DB      'U'
0880 BC                 DB      '<'+80H
0881 5D08               DW      LESS-4
                ;ULESS  DW      DOCOL,TDUP
                ;       DW      XORR,ZLESS
                ;       DW      ZBRAN,ULES1-$   ; IF
                ;       DW      DROP,ZLESS
                ;       DW      ZEQU
                ;       DW      BRAN,ULES2-$
                ;ULES1  DW      SUBB,ZLESS      ; ELSE
                ;ULES2  DW      SEMIS           ; ENDIF
                ;-------------------------------------------
0883 9508       ULESS   DW      $+2
0885 D1                 POP     D       ; (DE) <- (S1) = Y
0886 E1                 POP     H       ; (HL) <- (S2) = X
0887 CD3C08             CALL    SSUB    ; (HL) <- X - Y
088A 210000             LXI     H,0
088D D29108             JNC     ULES1
0890 2C                 INR     L
0891 C34401     ULES1   JMP     HPUSH
0894                    END
```

Wydruk 5

— jeżeli dopisane zostały nowe definicje, należy zmienić wartość słowa o adresie HEX 10C na adres pola NF ostatniego słowa w słowniku FORTH,

— na dysku należy zapisać zawartość pamięci od adresu 100_H do HERE.

Na zakończenie bieżącego odcinka zostanie zaprezentowana nowa wersja procedury U< napisanej w kodzie maszynowym (17 bajtów, bez nagłówka definicji — wydruk 5), a nie jak oryginalna — w języku FORTH (28 bajtów). Porównanie adresów nową procedurą trwa ok. 900 mikrosekund (mikroprocesor 8080, zegar 2 MHz), podczas gdy wykonanie starej wersji — sześć razy dłużej. W podane czasy wliczony jest czas pracy interpretera wewnętrznego („NEXT").

MICHAŁ CHOROSZUCHA
Warszawa

Wprawdzie zajęcia Akademii uległy oficjalnemu zakończeniu, to jednak zapraszamy Czytelników na wykład uzupełniający. O ile w pismach rozmaitego pokroju wprost zaroiło się od przeróżnych kursów programowania, to jednak znikoma liczba autorów wpaja swoim czytelnikom zasadę, że wiedza o tym, co realizuje taka czy inna instrukcja, to dopiero połowa (a może i mniej) umiejętności składającej się na sztukę programowania. O tym, że programowanie jest sztuką, przekonują się natychmiast wszyscy ci, którzy z jakichś względów zmuszeni są do wprowadzenia zmian w cudzych programach. O ile przy krótkich programach hołdowanie zasadzie: lepiej napisać własny program niż „przegryzać się" przez cudzy — może jeszcze dawać pewne rezultaty, to już przy tylko nieco dłuższych jest to praktycznie niemożliwe. Gorzej, gdy przychodzi „przegryzać się" przez ... własny program, tyle że napisany np. rok temu.

Krótki kurs Akademii mikroKLANU z pewnością nie jest w stanie wyczerpać całokształtu zagadnień związanych z umiejętnością „eleganckiego" programowania. Poniższe post scriptum poświęcone jest metodologii tworzenia programów średniej długości.

Post scriptum Akademii mikroKLANU czyli jak pisać duże programy

Tak się złożyło, że wszystkie odcinki Akademii mikroKLANU poświęcone były małym programom komputerowym. Aby zakończyć cykl, trzeba jeszcze wspomnieć o dużych programach. Wprawdzie większość użytkowników mikrokomputerów nigdy nie napisze naprawdę dużego programu (typu procesora tekstu lub systemu operacyjnego), ale nieco dłuższe programy zdarza się pisać prawie każdemu; wiele zastosowań mikrokomputera sprowadza się do napisania nietypowego programu. Programu takiego nie da się skopiować ani przerobić, trzeba go niestety napisać. Jak to zrobić skutecznie, postaram się pokazać w tym artykule. Jako przykład wybrałem program, który pisałem dla znajomego. Spełnia on opisane powyżej założenia, gdyż jest programem „średniodługim" (ok. 300 linii, ale nie będziemy ich wszystkich reprodukowali), a jednocześnie był pisany z myślą o bardzo prostym komputerze (ZX81) oraz o zupełnie niewprawnym użytkowniku, dla którego wiele wewnętrznych możliwości komputerka jest jeszcze tajemnicą. Program umożliwia sterowanie (poprzez specjalny interfejs) elektroniczną perkusją i służy do programowanego zmieniania granych melodii. Takie urządzenia oczywiście można już kupić (w cenie ok. 400 dol. USA), ale mój znajomy woli je zrobić sam (w Polsce ciągle łatwiej jest zbudować nawet skom-

plikowane urządzenie, niż zarobić pieniądze na jego kupno...) [1]).

Jesteśmy więc na starcie: postawiono problem, a my myślimy, jak może on zostać rozwiązany z wykorzystaniem mikrokomputera. Ta faza projektowania programu jest bardzo ważna. Można powiedzieć, że od właściwych decyzji na tym etapie zależy powodzenie całego projektu. A więc nie spieszmy się i dobrze przemyślmy cały problem. Zacznijmy od zanotowania, co program ma robić. W naszym przykładzie chcemy, by komputer zapamiętał 16 melodii, przy czym każda melodia składa się z 32 nut (część z nich może się powtarzać) Mamy do dyspozycji 12 instrumentów, ale prawdopodobnie nigdy wszystkie nie będą wykorzystane w jednej melodii. Należy więc zrobić zestawienie częstości wykorzystania poszczególnych instrumentów: tu pomoże nam ktoś doświadczony w sprawach muzycznych. Jeżeli zastanowimy się nad mnogością melodii oraz instrumentów, to dojdziemy do wniosku, że trzeba przyjąć jakiś sposób kodowania tak dużej liczby informacji. Jako wprawkę piszemy więc procedurę kodującą użycie poszczególnych instrumentów w danej nucie. Wydaje się, że kodowanie binarne jest tu optymalne. Oto proste procedury kodujące i dekodujące:

```
11000 REM PROCEDURA KODUJACA
11010 A=0
11020 FOR I=1 TO 12
11030 A=A+N(I)*2**I
11040 NEXT Y
11050 RETURN


10000 REM PROCEDURA DEKODUJACA
10010 IF A/2()INT(A/2) THEN GOTO 10100
10020 FOR I=12 TO 1 STEP -1
10030 N(I)=0
10040 B=A-2**I
10050 IF B<0 THEN GOTO 10080
10060 N(I)=1
10070 A=B
10080 NEXT I
10090 RETURN
10100 PRINT "BLAD DANYCH!"
10110 STOP
```

Ponieważ ZX81 ma niezbyt pewny system nagrywania danych, nasz sposób kodowania ma jeszcze jedną zaletę: umożliwa sprawdzenie, czy wczytane dane nie zawierają błędu. Oczywiście w praktyce nie musimy od razu pisać procedur w sposób formalny. Wystarczy, jeżeli wiemy, jak będziemy postępowali, czyli określić algorytm.

Problem, z którym przyszło nam się zmierzyć, byłby łatwiejszy do rozwiązania w języku bardziej sformalizowanym niż BASIC (np. PASCAL), ale i tu możemy zastosować elementy programowania strukturalnego. Trzeba jednak pamiętać o globalnym charakterze zmiennych w języku BASIC.[2]) Dlatego też niezbędne jest narysowanie dokładnego schematu logicznego programu. Należy zacząć od bardzo ogólnie sformułowanych zadań dla poszczególnych bloków funkcjonalnych, a potem schodzić do coraz niższych modułów. Takie postępowanie nazywa sie programowaniem zstępującym i należy je polecić wszystkim piszącym dłuższe programy. Prosze zwrócić uwagę, że po przyjęciu takiego stylu programowania samo zapisywanie programu jest już niewielkim, finalnym zadaniem. Ze względu na globalny charakter zmiennych w języku BASIC, procedury należy wprowadzać tylko wtedy, gdy wielokrotnie powtarza się konieczność realizacji identycznego zadania, tzn. gdy dany ciąg instrukcji może być wywoływany przez program główny wiele razy. W naszym przykładzie na pewno taka procedura będzie rysowanie graficzne taktu 32-nutowego:

Jest to procedura jedna z najważniejszych w tym konkretnym zagadnieniu. Ponieważ jest ona bardzo długa — a w tym tekście raczej chodzi nam o pokazanie metodologii programowania — więc nie będziemy jej publikować.

Analiza schematu logicznego wykazała, że w każdym przypadku, tzn. gdy tworzymy nową melodię, gdy uzupełniamy starą lub gdy ją odtwarzamy, wygodnie jest zobaczyć na ekranie komputera rozkład użycia instrumentów (znana zasada „rysunek wart jest 1000 słów" przyjmuje w przypadku komputerów formę „...wart jest 1024 bajty"!).

Specjalnie nie zamieszczam schematu logicznego przykładowego programu, gdyż chciałbym, by Czytelnicy sami zastanowili się, jak taki schemat mógłby wyglądać. Gdyby ktoś miał wątpliwości, to w dalszym ciągu pokazany będzie program główny, skąd Czytelnicy będą mogli zorientować się, jakie możliwości były rozpatrywane. Pamiętajmy, że dobry schemat logiczny to także połowa (a może więcej) dokumentacji programu. A dobrze napisany program musi być udokumentowany. Prawdopodobnie nasz program będziemy wielokrotnie przerabiali, a więc dokładny opis, co robi każdy moduł, jaka jest rola zmiennych itp., będzie niezbędny. Wreszcie, w naszych notatkach powinny się znaleźć uwagi na temat testowania programu i jego rezultatów. W ten sposób będziemy mogli łatwo usprawnić program.

Jeżeli już dokonaliśmy logicznego podziału programu na małe moduły, warto sprawdzić, czy są one zrozumiałe, to znaczy, czy nie trzeba będzie w nich zastosować instrukcji GO TO. Niestety, BASIC narzuca nam tu znaczne ograniczenie: na ogół nie dysponujemy instrukcją ELSE, a więc nie będzie można zrezygnować z instrukcji GO TO. Warto jednak (mówiliśmy już o tym w Akademii) stosować pewne chwyty programowe, by w jednym wyrażeniu arytmetycznym wykonywać wiele wariantów (przypominam program suwania kursorem).

Mając już określoną strukturę modułów, możemy teraz przystąpić do zapisywania programu. Dobrze jest zrobić sobie podręczną listę zmiennych i trzymać się ustalonej konwencji (pamiętamy, że w programach Akademii oznaczaliśmy zmienną operacyjną literą X, stałe literami A, B, C itd.). Oto jak zakodowaliśmy nasz program główny:

```
10 REM PROGRAM "PERKUSJA"
20 DIM M(16,32),M$(16,10),N$(12,10),N(12),T(16)
30 PRINT "MENU:"
40 PRINT ""
50 PRINT "NOWA MELODIA   1"
60 PRINT "UZUPELNIENIE   2"
70 PRINT "ODTWORZENIE    3"
80 PRINT ""
90 INPUT "PODAJ KOD:";P
100 CLS
110 PRINT "MELODIE:"
120 FOR I=1 TO 16
130 IF M$(I)="0 THEN GOTO 140
140 PRINT I;"-";M$(1)
150 NEXT I
160 PRINT ""
170 INPUT "WYBIERZ MELODIE:";N
180 IF P)1 THEN GOTO 190
190 INPUT "NAZWA MELODII:";M$(N)
200 CLS
210 GOSUB 5000
220 PRINT M$(N)
230 GOSUB 6000
240 GOSUB 7000
250 IF P$()"KONIEC" THEN GOTO 230
260 INPUT "POWTORZYC 16 NUT (T/N)";P$
270 IF P$="T" THEN GOSUB 8000
280 INPUT "TEMPO:";T
290 GOSUB 9000
300 INPUT "POPRAWKI (T/N)";P$
310 IF P$="T" THEN GOTO 20
320 END
```

Dla większej przejrzystości programu oznaczyliśmy kodowaną melodię jako M, nazwy kodowanych melodii zapamiętamy w wektorze MS, nazwy instrumentów zapamiętamy w wektorze NS. Analiza problemu wykazała, że zamiast nut będziemy wyświetlali symbol danego instrumentu. Tak więc prócz nazw instrumentów dobrze jest pamiętać ich litery kodujące:

```
5000 REM PROCEDURA LADUJACA NAZWY INSTRUMENTOW
5010 N$(1)="S-BAS DRUM"
5020 N$(2)="T-TAM TAM"
5030 REM ITD.
5100 RETURN
```

Procedura 5000 może też wyświetlać pełne nazwy instrumentów wtedy, gdy użytkownik zapomni czym dysponuje. Takie podwójne wykorzystanie danych bardzo skraca program — ZX81 ma małą pamięć!

Chciałbym teraz zwrócić uwagę Czytelników na pewną trudność, jaką można napotkać w trakcie pisania programu. Otóż z podziału na moduły wynikło, że dwa bloki będą wywoływane po sobie (linie 220 i 230, gdzie procedura rysująca obrazek taktu po-

[1]) Nie zgadzam się z autorem. Mikrokomputer łatwiej i taniej jest kupić, natomiast zarobienie na niego to zupełnie inna sprawa, faktycznie w Polsce niełatwa. Trochę inaczej wygląda kwestia urządzeń specjalizowanych, które produkowane są w małych seriach. Rzeczywiście często opłaca się zrobić je samemu, tak jak urządzenie wspomniane w artykule (przyp. AJP)

[2]) Uwaga dla nowicjuszy: zmienna globalna to taka, która jest identycznie rozumiana w całym programie; w niektórych językach programowania można wprowadzić zmienne lokalne dla danej procedury i w ten sposób uniknąć błędów wynikających z przypadkowej zbieżności nazw zmiennych stosowanych w różnych procedurach (przyp. AJP)

przedza procedurę wczytującą kod instrumentu oraz pytającą, czy już koniec poprawek):

```
7000 REM PROCEDURA WCZYTUJACA KOD INSTRUMENTU
7010 REM DOKONUJE POPRAWEK W GRAFICE
7020 REM KORZYSTA Z PROCEDURY 11000
7100 RETURN
```

Można oczywiście zapytać, dlaczego nie połączyliśmy tych procedur. Ano, gdybyśmy chcieli zmodyfikować nasz program (a nastąpi to z całą pewnością!), wygodniej będzie mieć te procedury rozdzielone. Podobny argument odnosi się do procedury powielającej 16 nut (takt równo podzielony). Gdybyśmy nie zdefiniowali tej procedury, należałoby stosować skok bezwarunkowy, co bardzo zmniejsza przejrzystość programu (linia 260). A tak piszemy prostą procedurę-moduł:

```
8000 REM PROCEDURA POWIELENIE 16 NUT
8010 FOR I=1 TO 16
8020 M(N,I+16)=M(N,I)
8030 NEXT I
8040 RETURN
```

Wreszcie dochodzimy do celu: trzeba już tylko napisać właściwą procedurę sterującą interfejsu — najlepiej zrobić to w języku wewnętrznym, gdyż BASIC ZX81 jest stosunkowo wolny, a użytkownik może zażądać szybkiego grania melodii (nie możemy łudzić się, że nasza perkusja będzie wykorzystywana tylko do podawania taktu bardzo wolnego poloneza...).

```
9000 REM PROCEDURA W JEZYKU WEWNETRZNYM
9010 REM STERUJACA ELEKTRONICZNA PERKUSJA
9100 RETURN
```

Oczywiście wszystkie procedury przetestowaliśmy osobno i dopiero po upewnieniu się co do ich prawidłowości oraz poprawności operacji łączymy je w jeden, długi program.

Na zakończenie kilka uwag praktycznych, choć osobiście wierzę, że nic nie jest w stanie zastąpić doświadczenia (proszę Czytelników, by jako małą wprawkę spróbowali napisać program elektronicznej perkusji na swój, wybrany komputer!).

Przede wszystkim nie należy bać się pisania modułów od nowa. Jeżeli mamy kłopoty, jeżeli program wygląda nieelegancko lub jest niezbyt przejrzysty, to należy siąść i napisać go po raz drugi. Być może należy nawet cofnąć się do wcześniejszego etapu, tzn. do planowania i dobrze przemyśleć, czy problem postawiliśmy we właściwy sposób. Zwykle dobrze „rozgryziony" problem koduje się sam, natomiast jeżeli mamy wątpliwości, to lepiej jeszcze raz przeanalizować sytuację! Po drugie nie należy uciekać od testów. Tylko wszechstronne sprawdzenie programu (także przez przyszłego użytkownika!) zapewni nam dobrą jakość produktu finalnego. W naszym przykładzie dzięki modułowemu charakterowi programu oraz konsekwentnej regule użycia zmiennych udało się wprowadzić — już po zakończeniu pisania — wiele drobnych zmian, które wydały się użytkownikowi konieczne dla łatwiejszego operowania (np. wprowadzenie zmiennej liczby instrumentów). Dobra dokumentacja (czyli notatki, gdzie i jaka zmienna jest używana) pozwoliły na bezbolesne wprowadzanie kolejnych zmian, choć program ciągle się rozrastał. W pewnym momencie np. okazało się, że dotychczasowy podział na moduły należy zmienić dla przyspieszenia działania programu. I znowu przejrzysta dokumentacja ułatwi szybkie przeniesienie określonych operacji do programu głównego. Wszystkie tego rodzaju zmiany były po prostu nie do pomyślenia przed przetestowaniem programu. A wydawało się, że wszystkie możliwości zostały rozważone...

Wreszcie dochodzi się do momentu, gdy dalsze poprawianie programu jest ... bezcelowe — to znaczy czas poświęcony na szlifowanie drobiazgów nie opłaci się. Po prostu program będzie używany przez określony czas i wszystkie poprawki, które trwają dłużej niż czas używania programu, są nieekonomiczne (pamiętajmy, że program piszemy w celu wykonania zadania, a nie dla efektu estetycznego!). Wszystkim, którzy pasjonują się programowaniem, polecam maksymę: „Najlepszy program, to program działający"!

Zamykając ostatnie posiedzenie Akademii mikroKLANU życzę wszystkim Czytelnikom wielu działających programów!!

JAKUB TATARKIEWICZ

Żegnamy mikroKLAN na łamach INFORMATYKI

mikroKLAN towarzyszył Czytelnikom przez dwa roczniki INFORMATYKI. Przez ten czas przeżyliśmy prawdziwą „mikrorewolucję": mikrokomputery pojawiły się w Polsce nie tylko jako barwne fotografie z zagranicznych pism, ale trafiły nawet do prywatnych domów. Najważniejsze jednak, że stały się narzędziem pracy — wiele importowanych komputerów zostało zakupionych przez instytuty naukowe i wyższe uczelnie. Tańsze modele trafiły, na razie na prawach eksperymentu, do wybranych szkół średnich. Z wąskiej grupy hobbistów, do której początkowo adresowaliśmy mikroKLAN, wyrosła liczna rzesza użytkowników ZX SPECTRUM, COMMODORE C64, AMSTRADA, a ostatnio nawet MACINTOSHA i IBM-PC.

W tej sytuacji kontynuacja mikroKLANU jako wkładki do niskonakładowego, specjalistycznego pisma, które na domiar złego ukazuje się ostatnio z wielomiesięcznym opóźnieniem, nie ma większego sensu. Zeszyty INFORMATYKI stały się nawet przedmiotem spekulacji, a ograniczona objętość wkładki powodowała, że nie można było w niej omówić wielu ważnych zagadnień.

Z górą rok temu podjęliśmy starania o utworzenie osobnego pisma poświęconego zagadnieniom związanym z racjonalnym stosowaniem techniki mikrokomputerowej. Przez długi okres nikt nie chciał uwierzyć (oczywiście z wyjątkiem Czytelników mikroKLANU, którzy zarzucali nas listami, nie mówiąc o telefonach), że takie pismo jest potrzebne, że może być sprzedawane nawet w 100 tys. nakładzie.

Po przezwyciężeniu przeszkód, wynikających z deficytu papieru i możliwości druku, wydawnictwo SIGMA uzyskało zezwolenie na wydawanie w 1986 roku serii broszur o nazwie „Mikroklan".

Czytelników, którzy mając na uwadze trudności, jakie występowały z nabyciem INFORMATYKI, chcieliby zaprenumerować „Mikroklan" musimy na razie rozczarować: broszury — jak wszystkie wydawnictwa nieperiodyczne — nie zostaną objęte prenumeratą.

REDAKCJA

prowadzi
ANDRZEJ J. PIOTROWSKI
tel. dom.: 48-22-85

dokończenie ze str. 19

Dostępne są funkcje generatorów rozkładów: równomiernego, nierównomiernego, wykładniczego, Poissona, Erlanga, normalnego i Bernoulliego.

Struktura programu w PASCALSIMIE ma strukturę programu PASCALA z następującymi rozszerzeniami:
- definicje typów kończy specyfikacja typów obiektowych po słowie ENTITY
- po deklaracji zmiennych pascalowych deklaruje się zmienne typów pascalsimowych po słowie SIMVAR
- deklaracje procedur pomocniczych wykorzystywanych w procedurach zdarzeniowych muszą poprzedzać procedury zdarzeniowe
- program główny tworzy blok zaczynający się słowem MAIN i kończący się słowem END.

SYSTEM PASCSIM I TRANSLATOR

System PASCSIM, przeznaczony do tłumaczenia i wykonywania programów symulacyjnych w języku PASCALSIM, zaimplementowano dla systemu operacyjnego RIAD/OS przy wykorzystaniu kompilatora PASCAL 360.

Translacja programu w PASCALSIMIE jest realizowana dwustopniowo:
- translacja z PASCALSIMA na PASCAL
- translacja z PASCALA na język wewnętrzny.

W skład systemu PASCSIM wchodzi:
- translator języka PASCALSIM na PASCAL
- biblioteka pomocniczych procedur symulacyjnych
- zbiór skatalogowanych procedur sterujących tłumaczeniem i wykonywaniem programów symulacyjnych.

Użytkownik systemu PASCSIM uzyskuje wydruk programu źródłowego w PASCALSIMIE, wydruk programu pośredniego w PASCALU oraz wydruk wyników programu symulacyjnego. Wykrycie błędów jest sygnalizowane na każdym poziomie.

Translator PASCALSIMA zajmuje ok. 38 K słów pamięci, a biblioteka procedur symulacyjnych — ok. 14 K słów.

Translator języka PASCALSIM dokonuje analizy syntaktyczno-semantycznej programu pomijając analizę fragmentów pascalowych oraz generuje w PASCALU postać pośrednią programu symulacyjnego.

Powyższe funkcje realizuje w następujący sposób:
- rozszerza deklaracje typów użytkownika o deklaracje standardowych typów danych języka PASCALSIM definiujac rekordy elementów kolejki, elementów kalendarza, nagłówka kolejki, nagłówka kalendarza, danych statystycznych i obiektu tymczasowego, w oparciu o atrybuty obiektu zdefiniowane przez użytkownika oraz typy wskaźnikowe związane z pewnymi typami rekordowymi
- generuje deklaracje standardowych zmiennych niezbędnych do symulowania upływu czasu, zerowania statystyk i śledzenia pracy programu
- dołącza zapowiedzi standardowych procedur symulacyjnych z biblioteki
- generuje niektóre procedury realizujace instrukcje PASCALSIMA uzależnione od struktur danych programu symulacyjnego, tj. procedur inicjowania stanu modelu, zerowania zmiennych typów statystycznych i wyszukiwania w liście obiektu o określonej wartości określonego atrybutu
- generuje procedury dotyczące kalendarza zdarzeń, odpowiednio do zadeklarowanych procedur zdarzeniowych
- generuje instrukcje ustalania stanu początkowego zmiennych standardowych
- zmienia instrukcje języka PASCALSIMA i ich argumenty na wywołania odpowiednich procedur bibliotecznych PASCALA z właściwymi parametrami
- kopiuje do pliku wyjściowego instrukcje pascalowe
- sygnalizuje wykrycie błędu w instrukcji PASCALSIMA.

POMOCNICZE PROCEDURY SYMULACYJNE I PROCEDURY STERUJĄCE

Biblioteka procedur pomocniczych zawiera moduły ładowalne standardowych procedur symulacyjnych, niezależnych od rodzaju symulowanego problemu.

W skład biblioteki wchodzą różne warianty procedur realizujących instrukcje języka PASCALSIM, tj. procedury działania na listach dwukierunkowych dla obiektów jedno- i wielowariantowych, procedury dotyczące kalendarza zdarzeń, procedury obsługi zmiennych typów statystycznych oraz funkcje generatorów liczb pseudolosowych.

Procedury dla obiektów wielowariantowych są wykorzystywane wtedy, gdy w programie symulacyjnym zadeklarowano więcej niż jeden typ obiektów tymczasowych. Procedury obsługi zmiennych statystycznych, tj. procedury zerowania, aktualizacji i drukowania zmiennych statystycznych z histogramem są dołączane wtedy, gdy dla jakiejkolwiek zmiennej danego typu statystycznego zadeklarowano potrzebę wyznaczania histogramu.

Do wynikowego programu symulacyjnego są dołączane w fazie łączenia tylko wykorzystywane w nim moduły. Biblioteka może być używana niezależnie od translatora PASCALSIMA do programowania zadań nie związanych z symulacją, a wykorzystujących funkcje w niej zawarte.

W celu ułatwienia uruchamiania programów w systemie operacyjnym OS/JS opracowano kilka procedur sterujących, odpowiednich dla różnych etapów uruchamiania programu.

Procedury te umożliwiają wykonanie następujących zadań:
- analizy syntaktycznej programu źródłowego z wyprowadzaniem wydruku programu źródłowego i sygnalizacją błędów
- kompilacji programu w PASCALSIMIE na program pośredni w postaci modułu pascalowego i kompilacji pro-

```
 0 PROGRAM SKLEPSIM (INPUT, OUTPUT);
 1 CONST      N=5;
 2 TYPE       OBIEKT=RECORD
 3              NRK:INTEGER; (*NR KASY*)
 4              CW:INTEGER; (*CZAS WEJSCIA*)
 5              CRF:REAL; (*CZAS ROZP FAZY*)
 6              CO :REAL; (*CZAS OCZ.*)
 7            END;
 8 ENTITY     OBIEKT;
 9 VAR      LAMBDA:INTEGER; (*PAR. ROZKL. CZASU POM. ZGL.*)
10          MIZAK:INTEGER; (*PAR. ROZKL. CZASU FAZY ZAKUPU*)
11          MIPLAC:INTEGER; (*PARI, ROZKL. CZASU FAZY PLACENIA*)
12          TKS:INTEGER; (*CZAS KONCA SYMULACJI*)
13          M: INTEGER; (*LICZBA KOSZY*)
14          MW:INTEGER; (*LICZBA KOSZY WOLNYCH*)
15          NW:INTEGER; (*LICZBA KAS WOLNYCH*)
16          LKO:INTEGER; (*LICZBA KLIENTOW OBSLUZONYCH*)
17          J:INTEGER; (*NUMER KASY O NAJKROTSZEJ KOLEJCE*)
18 SIMVAR  KLIENT:OBIEKT; (*REF DO OBIEKTU TYMCZ.*)
19          KKOSZ: QUEUE; (*KOL. DO KOSZY*)
20          KKAS:ARRAY[1..N] OF QUEUE; (*KOL. DO KAS*)
21          DKKOSZ:STAT; (*DL. KOL. DO KOSZY*)
22          DKKAS:ARRAY[1.. N] OF STAT; (*DL.KOL.DO. KAS*)
23          WIKAS:ARRAY[1..N] OF STAT; (*WSP. WYK.KAS*)
24          ZKOSZ:STAT; (*ZAJETOSC KOSZY*)
25          ZKAS:STAT; (*ZAJETOSC KAS*)
26          TOB:RAND; (*CZAS POBYTU*)
27          TOCZ:RAND; (*CZAS OCZEKIWANIA*)
28
29 PROCEDURE MIN;(*WYZNACZA NUMER KASY J   NAJKROTSZEJ KOLEJCE*)
30 VAR L:INTEGER;
31 BEGIN
32          J:=1;
33          FOR L:=1 TO N DO
34          IF SIZEOF(KKAS[J]) > SIZEOF(KKAS[L])
35          THEN J:=L;
36          END;
37—145 (*INNE PROCEDURY)
146 MAIN (*PROGRAM GLOWNY*)
147       LKO:=0;
148       READ (TKS,M,LAMBDA,MIZAK,MIPLAC);
149       NW:=N;
150       MW:=M;
151       INITIALIZE;
152       TRACE:=FALSE;
153       CREATE KLIENT;
154       PLAN ZGLOSZ FOR KLIENT AT GENEXP(LAMBDA, XX);
155       WHILE TIME<=TKS DO
156       SCAN;
157       RAPORT
158 END.
```

gramu w PASCALU z wyprowadzaniem programu źródłowego w PASCALSIMIE i programu pośredniego w PASCALU

- kompilacji pascalsimowej i pascalowej z drukowaniem programu źródłowego i pośredniego oraz łączeniem modułu wynikowego z procedurami z biblioteki i umieszczeniem modułu binarnego w zbiorze użytkownika w celu wielokrotnego wykonania programu
- kompilacji pascalsimowej i pascalowej z łączeniem i uruchomieniem programu symulacyjnego.

*

Przedstawiony język symulacyjny PASCALSIM ułatwia programowanie symulacji, zmniejszając wielkość i złożoność struktur danych użytkownika i rozmiary programu źródłowego, a tym samym czas i koszty uruchamiania programu.

Uzyskiwana postać programu symulacyjnego w PASCALU pozwala wnioskować o sposobie realizacji standardowych funkcji symulacyjnych w tym języku oraz o rozmiarach programu przy programowaniu symulacji w PASCALU.

Przytoczony fragment programu symulacji sklepu samoobsługowego wyposażonego w N-kas i M-koszy ilustruje sposób deklaracji zmiennych typów PASCALSIMOWYCH, postać instrukcji, strukturę programu i sposób drukowania wyników dla zmiennych statystycznych. Program ten, zawierający 158 wierszy, jest tłumaczony na 301 wierszy programu w PASCALU, nie uwzględniając procedur pomocniczych dołączanych z biblioteki.

universitas

STUDENCKA SPÓŁDZIELNIA PRACY

Oferuje usługi z zakresu projektowania
i programowania systemów informatycznych
oraz obsługi eksploatacyjnej
i technicznej komputerów

Zlecone prace wykonują studenci
wyższych lat studiów
pod nadzorem i przy udziale pracowników
warszawskich wyższych uczelni.

**Zgłoszenia w godz. 7.30—16.00 przyjmuje
Zakład Usług Informatycznych SPP UNIVERSITAS
Warszawa, ul. Nowowiejska 37, p. 2, tel. 25-56-28**

EO/132/K/85

amepol

AMEPOL — Przedsiębiorstwo Zagraniczne w Polsce oferuje do sprzedaży po nowych atrakcyjnych cenach niezawodne

PÓŁPRZEWODNIKOWE PAMIĘCI OPERACYJNE do następujących komputerów i minikomputerów:

- **ODRA 1305 o pojemności 64K, 128K, 192K, 256K, 512K,**
- **ODRA 1325 o pojemności 64K i 128K,**
- **RIAD R-32 o pojemności 256KB, 512KB, 768KB, 1MB i 2MB,**
- **MERA 400 o pojemności 64K, 128K, 256K, 512K,**
- **MERA 9150 o pojemności 64K,**
- **NOVA 1200 o pojemności 64K,**
- **PDP 11 o pojemności 64K i 128K,**
- **SM 4 o pojemności 64K i 128K.**

Pamięci półprzewodnikowe P.Z. AMEPOL skonstruowane są w oparciu o elementy DRAM VLSI produkcji zachodniej, co zapewnia dużą szybkość działania i znikomy pobór mocy z jednego źródła zasilania. Pamięci mogą współpracować z pamięciami ferrytowymi lub pracować samodzielnie.

Do końca bieżącego roku wdrożone będą do produkcji kontrolery dyskowe i procesory komunikacyjne. Przewidujemy również produkcję innych urządzeń informatyki.

Oferujemy krótkie terminy dostaw i instalacji. Każdy wyrób posiada roczną gwarancję i pełny serwis pogwarancyjny.

P.Z. AMEPOL uprzejmie prosi wszystkich zainteresowanych naszymi wyrobami o kontakty listowne, telefoniczne lub teleksowe.

**Nasz adres:
Plac Żelaznej Bramy 1
00-136 Warszawa**

PRZEDSIĘBIORSTWO ZAGRANICZNE

amepol

**Zakład Elektroniki i Aparatury Medycznej
Telefon: 20-34-75, 20-45-05
Teleks: 812539 apol pl**

EO/377/K/85

Skomputeryzowane systemy zarządzania w górnictwie węgla kamiennego

Prace badawcze i projektowe dotyczące zastosowania komputerów na potrzeby zarządzania w przemyśle węgla kamiennego, zainicjowane początkowo w Głównym Instytucie Górnictwa, od 1975 r. są systematycznie rozwijane w Centralnym Ośrodku Informatyki Górnictwa. Prace te pozwoliły szeroko rozpowszechnić w zarządzaniu przedsiębiorstwami branży węgla kamiennego systemy i metody informatyczne. Ze względu na zmieniające się potrzeby użytkowników oraz postęp w zakresie rozwiązań projektowo-programowych, eksploatowane są obecnie z reguły drugie, a niekiedy nawet już trzecie generacje tych systemów. Wiele z nich zaadaptowano i wdrożono w innych branżach resortu.

Prace badawcze prowadzone są zgodnie z koncepcją modelu kompleksowej komputeryzacji zarządzania, który ukierunkowany jest na usprawnienie rozrachunku gospodarczego i analiz oraz metody i technikę perspektywicznego, 5-letniego, rocznego i operatywnego planowania produkcji i inwestycji. Model ten składa się ze wzajemnie powiązanych systemów informatycznych, obejmujących całokształt produkcyjnej, inwestycyjnej, pomocniczej i usługowej działalności górnictwa węgla kamiennego. Wyodrębniono w nim 19 systemów analityczno-rozliczeniowych oraz 15 systemów planistycznych.

Systemy analityczno-rozliczeniowe dotyczą:

- zbiorczej oceny techniki i technologii w zakresie rozliczenia procesów produkcyjnych w kopalniach węgla kamiennego oraz działalności inwestycyjnej przemysłu węglowego (dwa systemy)
- ewidencji i analizy informacji dyspozytorskich, gospodarki zasobami węgla, środkami trwałymi, materiałowej, zatrudnieniowo-płacowej i księgowo-finansowej, kalkulacji kosztów własnych oraz ewidencji i rozliczania zbytu węgla (osiem systemów, tzw. podstawowych)
- oceny efektywności osiąganej w poszczególnych jednostkach działalności usługowej, zaplecza naukowo-badawczego itp. (dziewięć systemów).

Systemy planistyczne podzielono na trzy grupy:

- grupa I — planowania produkcji i inwestycji w kopalniach w okresach od jednego roku do kilkudziesięciu lat (dwa systemy)
- grupa II — operatywnego planowania produkcji i robót przygotowawczych w kopalniach, centralne wieloletnie planowanie podstawowych inwestycji i potencjału wykonawstwa inwestycyjnego oraz krótkookresowe planowanie wykonawstwa inwestycyjnego (cztery systemy)
- grupa III — planowania działalności usługowej.

Do chwili obecnej wdrożono całość (19) systemów analityczno-rozliczeniowych oraz sześć systemów planistycznych. Oto krótka charakterystyka podstawowych systemów.

Komputeryzacja metod ewidencji, rozliczania i oceny efektywności procesów produkcyjnych oraz gospodarki złożem

System rozliczania i analizy procesów produkcyjnych kopalń węgla kamiennego — **system IOS** — wiąże usprawnienie wewnętrznego rozrachunku gospodarczego kopalń ze stosowanymi technologiami produkcji. Jest on ściśle zintegrowany z innymi dziedzinowymi systemami zarządzania, dotyczącymi gospodarki materiałowej, energetycznej, zatrudnieniowo-płacowej, gospodarki wyposażeniem kopalń oraz rozliczeń księgowo-finansowych.

Dzięki wprowadzaniu jednolitych zasad ewidencjonowania i rozliczania zaszłości gospodarczych i kosztów wg układu stanowisk, oddziałów, rejonów rozliczeniowych i przodków powstały warunki tworzenia w komputerze banku informacji, zapewniającego ścisłe powiązanie między techniką, technologią i warunkami górniczo-geologicznymi a kosztami produkcji. Umożliwia to wykonywanie analiz procesu produkcyjnego w różnych przekrojach analitycznych (przodki, rejony, oddziały, ciągi technologiczne, systemy eksploatacji, pokłady, poziomy kopalni itp.) oraz stosowanie różnych metod ekonometrycznych.

System IOS składa się obecnie z kilkudziesięciu odcinków i modułów, które zostały wdrożone we wszystkich kopalniach.

Celem usprawnienia gospodarki złożem opracowano system **I-GZOP**, który wykorzystując dane przygotowane przez służby geologiczne tworzy bank danych o złożu. Umożliwia to wykonywanie analiz orz obowiązującej sprawozdawczości w zakresie stanu i zmian zasobów. System ten funkcjonuje we wszystkich kopalniach węgla kamiennego. Obecnie opracowuje się nową generację systemu gospodarki złożem, w którym wprowadzono m. in. automatyczne sporządzanie dokumentacji złóż. Rozwiązuje on zagadnienia automatycznego klasyfikowania i szacowania zasobów z wykorzystaniem metod geostatystycznych. Istotny postęp prac osiągnięto w zakresie automatycznego kreślenia izoliniowych map geologicznych, kart otworów wiertniczych i profili geologicznych.

Komputeryzacja działalności inwestycyjnej

Ze względu na wielomiliardową wartość inwestycji oraz stałą potrzebę odtwarzania zdolności produkcyjnych kopalń w miarę wybierania złóż — zarządzanie działalnością inwestycyjną w przemyśle węglowym jest problemem szczególnie złożonym. W dziedzinie tej skomputeryzowano:

- podstawowe rozliczenia i analizy realizacji procesu inwestycyjnego dotyczące robót budowlano-montażowych oraz całkowitych nakładów na obiekty i zadania inwestycyjne. Realizuje to podsystem ISB, obsługujący wszystkie szczeble służb inwestorskich w przemyśle węglowym (eksploatowany w 170 jednostkach organizacyjnych resortu)
- ewidencję, rozliczanie i analizę wyników produkcji podstawowej w przedsiębiorstwach budowlano-montażowych oraz zakładach remontowo-budowlanych w zakresie przerobów rzeczowych i finansowych, kosztów w układzie kalkulacyjnym, asortymentów robót oraz wyników działalności poszczególnych placów budów i oddziałów. Realizuje to podsystem ISB-B, eksploatowany we wszystkich (26) przedsiębiorstwach budowlano-montażowych oraz we wszystkich zakładach remontowo-budowlanych resortu
- ewidencję, rozliczanie i analizę wyników produkcji podstawowej w przedsiębiorstwach robót górniczych w zakresie przerobów rzeczowych i finansowych, normowania pracy, kosztów w układzie kalkulacyjnym, asortymentów robót oraz wyników działalności oddziałów. Realizuje to podsystem ISB-G, eksploatowany we wszystkich (10) zakładach robót górniczych
- bilansowanie i planowanie realizacji robót budowlano-montażowych (podsystem SOPR-BM1) oraz planowanie i rozliczanie zużycia materiałów na robotach górniczych w oparciu o normy techniczne materiałów (podsystem SOPR-BG2)
- bank informacji o realizacji procesu inwestycyjnego (podsystem ISB-P), który udostępniono resortowym biurom projektowym.

Systemy i metody informatyczne umożliwiły nie tylko udoskonalenie i pogłębienie dotychczasowych rozliczeń procesu inwestycyjnego w kopalniach oraz działalności przedsiębiorstw wykonawstwa inwestycyjnego, ale również wpływały na podniesienie poziomu organizacyjnego tych jednostek.

W zakresie komputeryzacji zarządzania działalnością inwestycyjną dąży się do osiągnięcia integracji procesu pla-

nowania, kontroli realizacji i rozliczania inwestycji. Zamierza się przystąpić do komputeryzacji operatywnego kierowania realizacją procesu inwestycyjnego przy wykorzystaniu minikomputerów.

Komputeryzacja gospodarki materiałowej

Celem usprawnienia kierowania i zarządzania gospodarką materiałową w poszczególnych jednostkach organizacyjnych oraz w całym przemyśle węgla kamiennego, opracowano cztery systemy analityczno-rozliczeniowe i dwa systemy planistyczne, które zapewniają pełną ewidencję obrotu magazynowego materiałów, części zamiennych i przedmiotów nietrwałych wraz z obowiązującą sprawozdawczością księgową i zaopatrzeniową, stwarzają podstawy do planowania zaopatrzenia materiałowego, dostarczają szeroką gamę rozliczeń i analiz oraz prowadzą podstawowy zakres operatywnej kontroli stanu gospodarki materiałowej.

Systemy te obejmują następujące trzy podstawowe sfery gospodarowania materiałami:

- w sferze zaopatrzenia — planowanie zaopatrzenia materiałowego, prognozowanie zużycia części zamiennych do maszyn i urządzeń górniczych, ewidencję i kontrolę zamówień materiałowych wraz z sygnalizacją zamówień o zagrożonej realizacji, kontrolę wykorzystania limitów zakupu materiałów, rozliczanie dostaw materiałowych (wraz z sygnalizacją materiałów w drodze i dostaw niefakturowanych) oraz prowadzenie banków informacji o poszukiwanych materiałach i częściach zamiennych
- w sferze magazynowania — księgową ewidencję obrotu magazynowego materiałów, kontrolę rozchodu materiałów limitowanych, ustalanie norm zapasów materiałowych dla grup i asortymentów, kontrolę i analizę zapasów materiałowych wraz z sygnalizacją zapasów nieprawidłowych (zbędnych i nadmiernych), sporządzanie sprawozdawczości zaopatrzeniowej resortowej i ogólnopaństwowej GUS, przeszacowywanie zapasów materiałowych w wyniku zmian cen, analizę kształtowania się kosztów materiałowych przedsiębiorstw w warunkach stosowania ruchomych cen zakupu, kontrolę gospodarki odpadami i surowcami wtórnymi oraz kontrolę zagospodarowania zapasów nieprawidłowych
- w sferze zużycia — ewidencję i rozliczanie materiałów zużywanych w procesie inwestycyjnym kopalń, rozliczanie zużycia materiałów według zleceń produkcyjnych, ewidencję i kontrolę odzysku oraz zagospodarowania materiałów wielokrotnego użytku, a także ewidencję przedmiotów nietrwałych w użytkowaniu.

Systemami gospodarki materiałowej objęto 281 jednostek organizacyjnych resortu.

Komputeryzacja gospodarki majątkiem trwałym i wyposażeniem technicznym

W dziedzinie gospodarki maszynami i urządzeniami eksploatowane są następujące systemy ewidencyjno-rozliczeniowe:

- **I-EAST** — ilościowo wartościowa ewidencja i rozliczanie środków trwałych oraz sprawozdawczość i analiza gospodarki zasobami majątku trwałego (eksploatowany w ponad 400 jednostkach organizacyjnych)
- **I-CWN** — pogłębiona ilościowa ewidencja podstawowych maszyn i urządzeń górniczych wraz z kontrolą struktury zagospodarowania i analizą stopnia wykorzystania wyposażenia górniczego (eksploatowany we wszystkich kopalniach węgla kamiennego)
- **I-EAD** — ewidencja, rozliczanie i analiza czasu pracy ciągów technologicznych oraz awaryjności maszyn i urządzeń zainstalowanych na stanowiskach pracy (eksploatowany we wszystkich kopalniach węgla kamiennego)
- **SEZAM** — kontrola realizacji potwierdzonych dostaw maszyn, urządzeń i części zamiennych (eksploatowany w 40 jednostkach organizacyjnych)
- BANK maszyn i urządzeń zbędnych, zdatnych do dalszego zagospodarowania (eksploatowany w 200 jednostkach organizacyjnych).

Warto podkreślić, że system I-EAST zapewnia:
— obniżenie pracochłonności ewidencji analitycznej, prowadzonej w pionie głównego księgowego i działach technicznych
— pełne skomputeryzowanie obliczeń amortyzacji, księgowości środków trwałych i sprawozdawczości GUS
— udostępnianie kierownictwu wszystkich szczebli zarządzania sprawnych mechanizmów analizy i kontroli gospodarki majątkiem trwałym
— udostępnienie informacji o majątku trwałym, niezbędnych do funkcjonowania podstawowych systemów zarządzania w resorcie górnictwa.

System I-CWN umożliwia lepsze wykorzystanie wyposażenia górniczego poprzez zagospodarowanie rezerw i właściwą eksploatację maszyn i urządzeń górniczych. Rozliczeniami objęto m.in.: kombajny ścianowe i chodnikowe, obudowę zmechanizowaną, przenośniki taśmowe, przenośniki zgrzebłowe, pompy, lokomotywy dołowe. Wydawnictwa wynikowe systemu I-CWN stanowią obowiązującą sprawozdawczość resortową.

Dalsze prace nad rozwojem systemów i metod związanych z gospodarką majątkiem trwałym będą realizowane w następujących kierunkach:

- komputeryzowanie nowych zagadnień z zakresu ewidencji, analizy i kontroli gospodarki środkami trwałymi, głównie dla użytkowników szczebla podstawowego; w związku z tym przewiduje się m.in.: intensyfikację prac nad zagadnieniami dotyczącymi charakterystyk technicznych obiektów inwentarzowych
- rozwój metod i systemów związanych z obsługą operatywnych potrzeb kopalń w zakresie gospodarowania majątkiem trwałym i wyposażeniem technicznym; realizacja tych prac odbywać się będzie z zastosowaniem minikomputerów do bieżącego rejestrowania dokumentów obrotu składnikami majątkowymi oraz operatywnej aktualizacji kopalnianego banku informacji o stanie ilościowym i strukturze zagospodarowania podstawowych obiektów inwentarzowych, w szczególności maszyn i urządzeń; ten kierunek prac ma zapewnić poprawę wykorzystania majątku; przewiduje się również podjęcie prac mających na celu przeniesienie kartotek systemowych ze zbiorów taśmowych na dyski oraz zapewnienie użytkownikom zdalnego dostępu do zbiorów systemu
- modernizacja algorytmów eksploatowanych na komputerach JS EMC, dostosowanych do wielodostępnej obsługi użytkowników, korzystania z techniki dyskowej oraz standardowego oprogramowania (zwłaszcza w zakresie informacji wynikowych).

Komputeryzacja gospodarki zatrudnieniowo-płacowej

System obliczania, rozliczania oraz analizy zatrudnienia i wynagrodzeń — system I-ERW — składa się z pięciu podsystemów:

1) obliczania wynagrodzeń pracowników produkcyjnych — I-ERW.1
2) obliczania wynagrodzeń pracowników umysłowych — I-ERW.2
3) rozliczania i ewidencji księgowej wynagrodzeń — I-ERW.3
4) analiz zatrudnienia i wynagrodzeń — I-ERW.4
5) sprawozdawczości resortowej i państwowej — I-ERW.5.

Podsystemy te realizują następujące funkcje:

- wprowadzanie, kontrola oraz bieżąca i okresowa ewidencja danych o czasie przepracowanym, nieprzepracowanym płatnym, nieprzepracowanym niepłatnym, danych osobowych pracowników oraz zakodowanych reguł postępowania w skali całego przedsiębiorstwa, np. sumaryczne wielkości potrąceń
- obliczanie wynagrodzeń dla wszystkich pracowników za czas przepracowany i nieprzepracowany (urlopy, choroby) oraz automatyczne przygotowanie wypłaty wynagrodzeń; z uwzględnieniem rozbicia na dowolną liczbę miejsc płatniczych, drukowanie indywidualnych odcinków zarobkowych, sporządzanie zapotrzebowań gotówki z banku wraz ze specyfikacją banknotów o potrzebnym nominale, rozliczanie potrąceń od wynagrodzeń, pracowniczej kasy zapomogowo-pożyczkowej i wiele innych dokumentów związanych z wypłatą wynagrodzeń
- automatyczne informowanie właściwych służb kopalnianych o wielkości i strukturze czasu pracy oraz wynagrodzenia każdego pracownika (bieżąco oraz narastająco); automatyczne opracowywanie zaświadczeń dla pracowników o wysokości wynagrodzeń (dla obliczeń rentowych i emerytalnych, szkół, przedszkoli itp.)
- rozliczanie i ewidencja księgowa kosztów wynagrodzeń w obowiązujących przekrojach (stanowiska kosztów, zlecenia, oddziały, konta syntetyczne i analityczne itp.); opracowywanie określonych przepisami zestawień kosztowych

- opracowywanie dla przedsiębiorstw oraz ministerstwa analiz zatrudnienia i wynagrodzeń, obejmujących rozkład przepracowanych dniówek oraz wysokości wynagrodzeń w poszczególnych grupach zawodowych, powiązanie wysokości i struktury wynagrodzeń z warunkami górniczo-technologicznymi; opracowywanie zależności ekonometrycznych dotyczących porównania wskaźników efektywności produkcji (wielkość wydobycia, pracochłonność, wydajność itp.) z wysokością i strukturą wynagrodzeń
- opracowywanie statystyk i sprawozdawczości zatrudnieniowo-płacowej dla wszystkich szczebli organizacyjnych resortu.

Podsystemy gospodarki zatrudnieniowo-płacowej wdrożono we wszystkich kopalniach węgla kamiennego i jednostkach budownictwa górniczego oraz w wielu innych przedsiębiorstwach.

Przykładowo, system I-ERW dostosowano do potrzeb kopalń węgla brunatnego i wdrożono w kopalni „Bełchatów". Prowadzone są również prace nad przystosowaniem systemu do potrzeb energetyki (wdrożenie pilotujące podsystemu I-ERW.1 w elektrowni „Bełchatów").

Prowadzone są w tej dziedzinie również prace zmierzające do zastosowania minikomputerów oraz wielodostępu.

Komputeryzacja rozliczeń kosztów, gospodarki księgowo-finansowej oraz zbytu węgla

System ewidencji i rozliczeń księgowych I-ERK umożliwia prowadzenie ewidencji i analizę obrotów, kont księgowych, rozliczeń z pracownikami, rozliczeń finansowych działalności inwestycyjnej oraz sporządzanie podstawowej sprawozdawczości dla jednostek organizacyjnych różnych szczebii. Dla zapewnienia bieżącego dostępu do gromadzonych informacji zastosowano w systemie sprzęt minikomputerowy, pozwalający na wstępne przetwarzanie danych bezpośrednio w przedsiębiorstwie. W tym celu opracowano pakiet programów użytkowych dla minikomputerów PSPD 90 oraz MERA 9150, który zapewnia przeglądanie na monitorze ekranowym oraz drukowanie informacji o obrotach i saldach kont księgowych.

Przewiduje się dalszy rozwój systemu, który będzie uwzględniał:

- dostosowanie systemu do współpracy z nowym, jakościowo lepszym sprzętem informatycznym, zapewnienie bezpośredniej współpracy minikomputerów zainstalowanych w przedsiębiorstwach z komputerami zainstalowanymi w COIG oraz udostępnienie poprzez teletransmisję bazy danych dla jednostek organizacyjnych szczebla nadrzędnego
- rozszerzenie zakresu funkcji systemu, mające na celu automatyzację innych pracochłonnych operacji ewidencyjno-rozliczeniowych, takich jak rozliczenia bankowe, fakturowanie i analiza sprzedaży oraz sporządzanie doraźnych analiz o aktualnej sytuacji finansowej i możliwościach płatniczych przedsiębiorstwa.

System I-ERK jest systemem funkcjonującym nie tylko we wszystkich kopalniach węgla kamiennego, ale również w wielu innych przedsiębiorstwach resortu.

W ramach systemu ewidencji, rozliczania i kalkulacji kosztów I-PRP eksploatowane są cztery podsystemy:

- ewidencji i rozliczania kosztów — **I-PRP.1**
- analiz kosztów — **I-PRP.2**
- ilościowego rozliczania energii — **I-PRP.4**
- kalkulacji jednostkowej i sprawozdawczości — **I-PRP.5**

Podsystem I-PRP.1 pobiera informacje o kosztach z systemów analityczno-rozliczeniowych, takich jak: I-EAST, I-ZGM, I-ERW, I-ERK, I-PRP.4 oraz informacje niezbędne do kalkulacji kosztu własnego produkcji z dokumentów źródłowych systemu I-PRP.

W wyniku przetworzenia danych z systemów analityczno-rozliczeniowych oraz dokumentów źródłowych systemu I-PRP powstaje rozliczenie kosztów prostych i złożonych, a użytkownicy otrzymują takie dokumenty wynikowe, jak: rejestry kont, stanowisk i zleceń oraz polecenia księgowania wraz z arkuszami pomocniczymi, dotyczącymi szczegółowego rozliczenia (np. obiektów budowlanych). W ramach tego podsystemu tworzona jest również baza informacyjna o kosztach własnych kopalń.

W oparciu o dane z podsystemu I-PRP.1, uzupełnione informacjami z systemu I-ERK, podsystem I-PRP.2 emituje zestawienia wynikowe obrazujące strukturę rodzajową kosztu własnego sprzedaży węgla. W opracowaniu znajduje się analiza rozliczeń międzyokresowych, która ma być wdrożona w 1986 r.

Podsystem I-PRP.5 sporządza obowiązujące w resorcie statystyki W-Ks na temat kalkulacji kosztu węgla w kopalniach oraz poszerzone analizy dotyczące robocizny i materiałów.

W planach dalszego rozwoju systemu I-PRP zmierza się do opracowania podsystemów planowania kosztów (I-PRP.3) oraz rozliczania i kalkulacji kosztów dla przedsiębiorstwa o dowolnym profilu produkcyjnym, a także do opracowania następnych skomputeryzowanych metod analizy kosztów własnych (ze szczególnym uwzględnieniem analizy kosztów wytwarzania węgla) oraz doskonalenia informatycznych metod rozliczania i kalkulacji kosztów jednostkowych.

System rozliczania i analizy zbytu węgla I-CZW funkcjonuje w Centrali Zbytu Węgla, obejmując podstawowe rozliczenia finansowe oraz statystykę i analizę sprzedaży węgla. Rozliczenia są realizowane przez następujące podsystemy:

- rozliczeń dziennych zbytu węgla — **I-CZW.1**
- rozliczeń finansowych dostawców węgla — **I-CZW.2**
- rozliczeń finansowych odbiorców węgla — **I-CZW.3**
- rozliczeń eksportowych węgla — **I-CZW.4**
- kontroli realizacji zamówień — **I-CZW.5**
- statystyk zbytu węgla — **I-CZW.6**
- reklamacji jakościowych węgla — **I-CZW.7.**

Obecnie opracowuje się drugą generację tego systemu przeznaczoną do eksploatacji na komputerach Jednolitego Systemu, co umożliwi dalsze usprawnienie obsługi Centrali Zbytu Węgla przez zastosowanie teletransmisji i innych nowoczesnych rozwiązań informatycznych.

Systemy wspomagające procesy planowania produkcji i inwestycji

Spośród systemów planistycznych najbardziej zaawansowano prace dotyczące systemów planowania produkcji i inwestycji.

W ramach systemu perspektywicznego planowania budowy i rozwoju kopalń SPP, do oceny technicznych koncepcji rozwojowych poszczególnych kopalń oraz wyboru koncepcji najbardziej efektywnej w skali branży lub grupy kopalń służy pakiet programów SPP, którego istota polega na współdziałaniu metody symulowania w komputerze przyszłych procesów produkcyjnych i inwestycyjnych z ekonometrycznymi modelami prognostycznymi. Natomiast wyznaczanie zestawów koncepcji rozwojowych kopalń do planów perspektywicznych branży lub gwarectw przeprowadzane jest za pomocą pakietu programów SPP.2, który wykorzystuje do tego celu metodę programowania liniowego.

Dotychczasowe prace nad systemem SPP doprowadziły do upowszechnienia metody symulowania, realizowanej za pomocą pakietu programów SPP.1. W latach 1974—1984 pakiet SPP.1 zastosowano do analizy ponad sześćdziesięciu wariantów rozwojowych, opracowanych dla prawie trzydziestu kopalń — czynnych lub projektowanych.

Doświadczenia z zastosowań pakietu SPP.1 miały wpływ na dalsze prace nad skomputeryzowaną metodą oceny wariantów rozwojowych. Ważniejsze kierunki tych prac, to dalsza integracja pakietu SPP.1 z systemami analityczno-rozliczeniowymi (uwzględniająca m. in., problem jakości produkowanego węgla oraz prowadzenie różnorodnych badań ekonometrycznych procesu produkcyjnego w kopalniach), a także rozszerzanie zakresu analiz koncepcji projektowych, m. in. doskonalenie ekonometrycznych modeli prognostycznych, wykorzystywanych w pakiecie SPP.1.

Zagadnienia dotyczące optymalnego rozdziału zadań produkcyjnych, limitowanego wyposażenia wyboru przedsięwzięć modernizacyjnych w skali branży lub grupy kopalń na okresy pięcioletnie i roczne rozwiązywane są za pomocą skomputeryzowanego systemu planowania pięcioletniego i rocznego SPK.

Podstawowe funkcje systemu SPK realizowane są przez swoisty „przetarg" ofert planistycznych przygotowanych przez kopalnie, który jest rozstrzygany za pomocą metody programowania liniowego.

Rozwiązania informatyczne zastosowane w systemie SPK pozwalają prowadzić w komputerze analizy stabilności uzyskanych rozwiązań (planów produkcji) przy z góry założonych funkcjach celu. W systemie przewidziano również możliwość wyboru ofert planistycznych do optymalizacji wielokryteriowej.

W latach 1977—1983 system SPK był wykorzystany do analizy pięcioletnich i rocznych planów produkcji w piętnastu kopalniach węgla kamiennego.

W połowie lat siedemdziesiątych szerokie zastosowanie w kopalniach węgla kamiennego znalazła skomputeryzowana metoda operatywnego planowania robót przygotowawczych użyta w systemie SPO-RP. Na podstawie skomputeryzowanej analizy sieci robót przygotowawczych, wykonywanej za pomocą pakietu programów PERT-ICL, wyznacza się harmonogramy robót oraz określa zapotrzebowanie na środki niezbędne do realizacji tych harmonogramów. Duża częstotliwość wyznaczania takich harmonogramów przy jednoczesnej zmienności warunków górniczo-geologicznych w kopalniach uwypukliła wady przetwarzania wsadowego. Obecnie prowadzone są prace nad wspomaganiem operatywnego planowania produkcji z zastosowaniem minikomputerów. Przewiduje się, że prototypowe rozwiązania zostaną udostępnione kopalniom w 1986 r.

Z przeprowadzonych charakterystyk poszczególnych systemów informatycznych wynika, że model kompleksowej komputeryzacji zarządzania w części systemów analityczno-rozliczeniowych został w zasadzie zrealizowany. Rozwiązania systemów różnią się oczywiście liczbą oddzielnych modułów (za moduł uważa się rozwiązanie informatyczne mające tzw. samodzielną zdolność wdrożeniową) i zakresem upowszechnienia, który na ogół dotyczy całej branży węgla kamiennego, a w wielu przypadkach nawet wykracza poza branżę. Natomiast mniej zaawansowane są prace nad systemami planistycznymi. Taki stan rzeczy spowodowało głównie niewielkie zainteresowanie tego typu rozwiązaniami — może ze względu na zastosowanie w nich skomplikowanych metod matematycznych. Takie systemy nie znajdują jeszcze powszechnego uznania użytkowników.

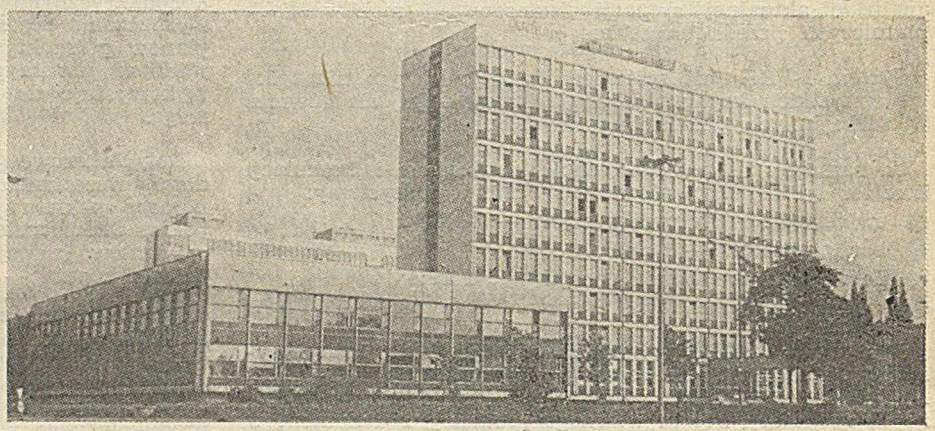
Oprócz wymienionych systemów dziedzinowych oraz metod szerokie zastosowanie w kopalniach i przedsiębiorstwach znalazły:

- banki różnych danych techniczno-ekonomicznych
- uniwersalne pakiety programów użytkowych, umożliwiające udostępnianie użytkownikom informacji na żądanie oraz sporządzanie analiz według nietypowych układów
- usprawnione metody dekretowania zaszłości gospodarczych i ich rozliczania oraz znacznie pogłębione i rozszerzone zakresy analiz i ocen procesów produkcyjnych w poszczególnych ogniwach technologicznych
- ujednolicone klasyfikacje, niezbędne do szczegółowego obserwowania i analizy procesu inwestycyjnego, systemu wynagrodzeń, gospodarki maszynami i urządzeniami itp. oraz ujednolicona ewidencja i kalkulacja kosztów, obliczania wynagrodzeń itp.
- skomputeryzowane systemy dotyczące prognozowania ekonomicznej efektywności określonych wariantów działania zmierzającego do utrzymania lub poprawy osiąganych wyników oraz optymalizacji decyzji przy założonych warunkach i kryteriach.

Wydawnictwa wynikowe uzyskiwane ze skomputeryzowanych systemów zarządzania stanowią obecnie rzeczywiste narzędzie pracy kadry kierowniczej i wielu służb kopalń i przedsiębiorstw, zwłaszcza niższego i średniego szczebla. Kadra kierownicza wyższego szczebla korzysta z systemów informatycznych raczej w sposób pośredni, co związane jest głównie ze stosowanym powszechnie trybem udostępniania informacji w postaci sformalizowanych, zbyt obszernych wydruków danych.

Jedną z przyczyn niepełnego wykorzystania możliwości tkwiących w skomputeryzowanych systemach jest eksploatowany sprzęt informatyczny. Systemy funkcjonują głównie w oparciu o komputery ODRA 1305, zainstalowane w Centrum Naukowo-Produkcyjnym COIG oraz w jego oddziałach. Większość systemów wykorzystuje odpowiednio zmodyfikowane dokumenty źródłowe sporządzane w kopalniach i przedsiębiorstwach. Informacje znajdujące się w tych dokumentach rejestrowane są za pomocą systemów MERA 9150 na taśmie magnetycznej, poddawane programowej kontroli, a następnie przetwarzane do postaci żądanej przez użytkowników. W trakcie kontroli kopalnie są angażowane również do oceny wiarygodności danych. Ten tryb przetwarzania, określany jako wsadowy, powoduje, że cykl przetwarzania nadmiernie wydłuża się, a gromadzone informacje mogą być wykorzystane na potrzeby operatywnego zarządzania w ograniczonym zakresie. Dlatego coraz szerzej stosuje się już rozproszone przetwarzanie danych. W tym celu w kopalniach instaluje się mikrokomputery, za pomocą których wprowadza się i gromadzi niezbędne informacje oraz wykorzystuje się je do zwiększenia operatywności działania danej komórki funkcjonalnej. Przykładowo do rozliczeń księgowo-finansowych w kopalniach zainstalowano już ok. 250 mikrokomputerów typu PSPD-90. Jest to zaledwie początek wprowadzania nowej technologii informatycznej do kopalń, lecz uznany obecnie za jeden z podstawowych kierunków prac prowadzonych w COIG. Drugim ważnym kierunkiem jest rozszerzenie stosowania wielodostępnej obsługi użytkowników oraz teletransmisji danych. Prace te wymagać będą jednak istotnej modernizacji eksploatowanego obecnie sprzętu informatycznego.

35 lat Centralnego Ośrodka Informatyki Górnictwa



Fot. 1. Gmach główny COIG

Początki stosowania środków informatycznych do prowadzenia prac ewidencyjno-rozliczeniowych w polskim górnictwie węglowym sięgają okresu międzywojennego, kiedy to spółki węglowe instalowały i eksploatowały pierwsze maszyny licząco-analityczne.

Bezpośrednio po II wojnie światowej, w wyniku przebudowy struktury gospodarczej kraju i związanej z tym zmiany systemu zarządzania, polski przemysł węglowy stanął w obliczu problemu dostosowania metod zarządzania do nowych warunków. Centrala Zbytu Produktów Przemysłu Węglowego w Katowicach zajmowała się wówczas dystrybucją produktów przemysłu węglowego i rozliczaniem ich dostaw, a zadania te realizowano początkowo przy użyciu przedwojennych zestawów maszyn licząco-analitycznych firmy HOLLERITH. W 1950 r. Centrala dysponowała już 7 zestawami.

Rosnące zapotrzebowanie na środki informatyczne spowodowało konieczność utworzenia samodzielnej, wyspecjalizowanej jednostki organizacyjnej. I tak 1 stycznia 1951 r. powstało Centralne Biuro Rozliczeń Przemysłu Węglowego (CBRPW). W 1952 r. zakupiono w ZSRR 10 zestawów maszyn licząco-analitycznych typu SAM, a pod koniec 1953 r. park maszynowy Biura umożliwił już realizację prac ewidencyjno-obliczeniowych dla większej liczby użytkowników.

Gwałtowna potrzeba rozszerzenia zakresu świadczonych usług informatycznych oraz związanego z tym doskonalenia technologii przetwarzania danych wymagała jednak wyposażenia CBRPW w bardziej wydajny sprzęt. Dotyczyło to także nowo powstałego oddziału w Wałbrzychu (1956), a potem kolejno uruchamianych oddziałów: we Wrocławiu (1965) i Rybniku (1969). Toteż w latach 1960—1965 — w ramach dalszej modernizacji parku maszynowego — zakupiono w firmie BULL maszyny skonstruowane już w oparciu o technikę elektroniczną (kalkulator elektroniczny lampowy Gamma 3-B, kalkulator tranzystorowy Gamma 172, czytnik-dziurkarkę ULP przystosowaną do odczytu 80-kolumnowych kart znakowanych oraz magnetyzator PM, dwa konwertery do przenoszenia danych z taśmy dziurkowanej na karty (PELLEROD), natomiast w ZSRR — trzy sortery elektroniczne (alfanumeryczne) SE-80-3 firmy SAM.

Przełomowym momentem dla przedsiębiorstwa było zainstalowanie w 1969 r. pierwszego w resorcie komputera ICL 1904E. Sprzęt ten całkowicie zmienił technologię przetwarzania danych oraz strukturę organizacyjną przedsiębiorstwa. Niezbędne stało się uzupełnienie sprzętu przez zakupienie w latach 1969—1972 w NRD 14 sorterów numerycznych SOEMTRON 434, a następnie w latach 1971—1974 — trzech perfosumatorów firmy ADDO oraz 30 dziurkarek alfanumerycznych i 16 sprawdzarek firmy SOEMTRON. Następnie z uwagi na coraz większe potrzeby użytkowników oraz efektywność stosowania perfosumatorów ADDO-X, zakupiono dalszych 175 maszyn tego typu. Nie zaspokoiło to jednak w pełni wszystkich potrzeb. Dlatego też w latach 1974—1975 zainstalowano 1022 perfosumatorów ASCOTA.

Istotną zmianą w historii rozwoju przedsiębiorstwa było przemianowanie go w 1972 r. na Centralny Ośrodek Informatyki Górnictwa i Energetyki (COIGiE), z przekazaniem mu funkcji nadzoru nad wszystkimi ośrodkami informatyki resortu. Rok później wyłączono jednak z COIGiE branże górnictwa naftowego i gazowego oraz energetyki, tworząc w nich samodzielne ośrodki.

1 stycznia 1975 r. ośrodek przeniesiono do nowego, specjalnie zaprojektowanego i wybudowanego gmachu, przy ul. Mikołowskiej 100 w Katowicach oraz zmieniono mu status przedsiębiorstwa państwowego na ośrodek badawczo-rozwojowy. Podstawą tych zmian było utworzenie pionu naukowo-badawczego, do którego włączono zespół pracowników byłego Ośrodka Naukowo-Badawczego Ekonomiki, Organizacji i Komputeryzacji Zarządzania Głównego Instytutu Górnictwa oraz pracowników komórek projektowania i programowania COIGiE. Równocześnie powołano Centrum Obliczeniowe COIGiE oraz przekazano zjednoczeniom przemysłu węglowego wspomniane trzy oddziały, które poprzednio wchodziły w skład przedsiębiorstwa.

W celu usprawnienia procesu przygotowania danych, realizowanego dotąd głównie przy zastosowaniu dziurkarek i sprawdzarek kart, zakupiono pierwsze wielostanowiskowe systemy rejestrowania danych na taśmach magnetycznych RC-3600 produkcji firmy REGENCENTRALEN (Dania). W latach 1976—1977 następuje istotne zwiększenie potencjału obliczeniowego w wyniku zainstalowania czterech komputerów ODRA 1305, w 1978 r. — komputera R 32. W latach 1979—1980 wzrosły również znacznie możliwości przygotowania maszynowych nośników informacji przez zakup dwóch wielostanowiskowych systemów rejestracji danych MERA 9150.

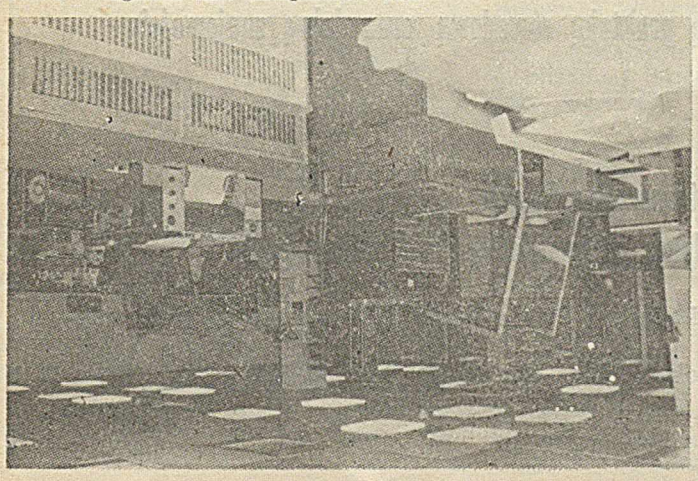
Po reorganizacji resortu, polegającej na wyłączeniu z niego energetyki, z dniem 31 marca 1976 r. Ośrodek otrzymał nazwę: Centralny Ośrodek Informatyki Górnictwa (COIG). W 1982 r. przywrócono Ośrodkowi status przedsiębiorstwa państwowego. Decyzją ministra górnictwa i energetyki 1 kwietnia 1984 r. przekazano do COIG ośrodki informatyczne funkcjonujące przy byłych zjednoczeniach kopalń węgla kamiennego, tworząc przedsiębiorstwo wielozakładowe z oddziałami terenowymi w Mysłowicach, Dąbrowie Górniczej, Katowicach, Bytomiu, Zabrzu, Rybniku i Wałbrzychu. Zadaniem tych oddziałów jest świadczenie usług informatycznych dla użytkowników zgrupowanych w siedmiu gwarectwach węglowych.

Mimo że przedsiębiorstwo przeszło już kilka reorganizacji, nie zmienił się charakter jego podstawowej działalności — jego służebna rola nie tylko wobec jednostek organizacyjnych resortu, ale również wielu jednostek pozaresortowych.

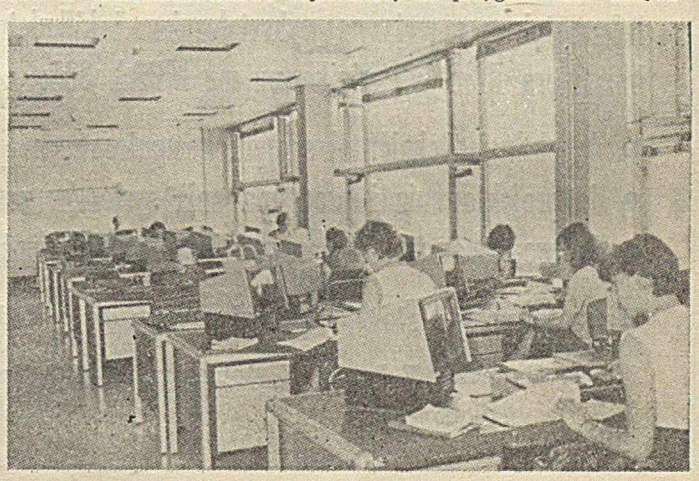
O pozycji, jaką w krajowej informatyce osiągnął COIG, zadecydowało jednak nie tylko bogate wyposażenie w sprzęt, zadecydowali ludzie, przede wszystkim ich wielkie zaangażowanie osobiste. Należy bowiem podkreślić, że szybki wzrost zapotrzebowania na usługi informatyczne odbywał się stale w sytuacji ciągłego deficytu wykwalifikowanych kadr.

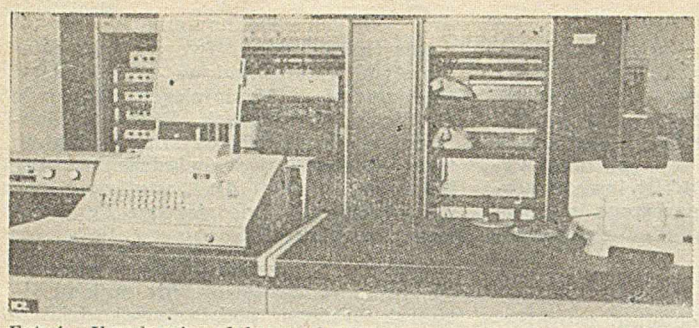
W pierwszych latach (1951—1952) funkcjonowania przedsiębiorstwa zatrudnienie wynosiło ok. 200 osób, a kadrę kierowniczą stanowiło zaledwie kilku specjalistów pracujących w tej dziedzinie jeszcze w okresie międzywojennym. Nowo przyjmowani pracownicy nabywali kwalifikacje wyłącznie w ramach tzw. szkolenia przywarsztatowego. Pierwszy kurs specjalistyczny, którego absolwenci otrzymali świadectwa, odbył się w 1951 r. i zapoczątkował systematyczne szkolenie wewnątrzzakładowe. Dopiero w 1959 r., z inicjatywy przedsiębiorstwa przy jednej z Zasadniczych Szkół Handlowych w Katowicach uruchomiono kierunek „Eksploatacja maszyn liczaco-analitycznych", a w 1960 r. w Technikum Ekonomicznym w Katowicach — kierunek „Mechanizacja prac obrachunkowych". Od 1968 r. zaczęto prowadzić szkolenie stacjonarne w Policealnym Studium Ekonomicznym przy Zespole Szkół Ekonomicznych w Katowicach (kierunek „Przetwarzanie danych"). W 1969 r. w szkole tej uruchomiono drugi kierunek informatyczny „Programowanie maszyn cyfrowych". Nauczanie w Policealnym Studium Ekonomicznym prowadzone jest nadal. Wielu absolwentów tych placówek zasiliło i nadal zasila załogę COIG.

Fot. 2. Fragment hali komputerów ICL

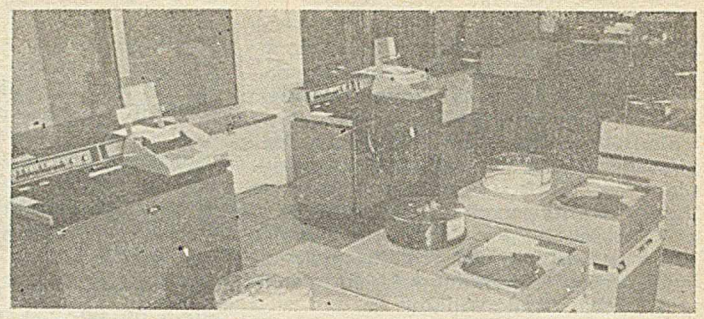


Fot. 3. Wielostanowiskowe rejestratory do przygotowania danych





Fot. 4. Urządzenia zdalnego dostępu



Fot. 5. Pamięci dyskowe

Począwszy od 1953 r. liczni pracownicy przedsiębiorstwa byli kierowani do ośrodków szkoleniowych krajowych i zagranicznych producentów urządzeń informatycznych, gdzie zdobywali niezbędne kwalifikacje zawodowe w zakresie obsługi technicznej sprzętu informatycznego oraz projektowania i programowania.

Wprowadzenie nowej technologii komputerowego przetwarzania danych wymagało nie tylko generalnego przekwalifikowania własnego personelu, ale również przeszkolenia użytkowników systemów informatycznych. W statucie COIGiE z 26 lipca 1972 r. po raz pierwszy znalazło się bowiem zadanie: *„organizowanie i prowadzenie szkolenia użytkowników systemów, analityków i projektantów, obsługi technicznej i operatorskiej"*. Do realizacji tego zadania w II połowie 1972 r. powołano w przedsiębiorstwie Dział Szkolenia Kadr Informatyki, przekształcony później w Resortowy Ośrodek Szkolenia Kadr Informatyki.

Aktualnie organizuje się rocznie przeszło 30 kursów różnych typów w kilku grupach tematycznych (propedeutyka informatyki, projektowanie i programowanie systemów, oprogramowanie standardowe, obsługa operatorska maszyn cyfrowych, systemy operacyjne itp.), a także specjalistyczne seminaria na temat rozwiązań informatycznych, ekonomicznych i organizacyjnych. Szkolenie to w ciągu roku obejmuje ok. 600 osób. Oprócz tego organizowane są praktyki zawodowe dla uczniów szkół średnich oraz studentów szkół wyższych, jak również staże zawodowe.

W 1973 r. w przedsiębiorstwie utworzono Dział Informacji Naukowej, Technicznej i Ekonomicznej, który rozpoczął wydawanie „Biuletynu Informatyki" oraz zeszytów problemowych z serii: „Technika obliczeniowa — Metody — Systemy — Programy".

W 1976 r. przyznano COIG funkcję resortowego ośrodka wiodącego w zakresie programowania, planowania i koordynacji działalności INTE. W związku z tym w latach 1977—1981 Dział INTE sukcesywnie rozszerzał działalność informacyjną. Wydawano „Informację Ekspresową", „Wiadomości z Zagranicznego Górnictwa Węglowego" oraz „Informacje o systemach informatycznych", zawierające m. in. tłumaczenia publikacji o najnowszych osiągnięciach zagranicznych. Obsługiwano także konferencje naukowe oraz współpracowano przy realizacji programu naukowego krajów socjalistycznych — INTERASUUGOL.

Od 1980 r. wydawane są zeszyty „Prace COIG", zawierające prezentację wyników prac naukowo-badawczych prowadzonych od 1976 r. w trzech problemach resortowych.

Prace naukowo-badawcze i wdrożeniowe dotyczące komputeryzacji zarządzania odznaczają się kompleksowością traktowania problemów zarządzania, etapowością realizacji oraz ścisłym powiązaniem doraźnych potrzeb kopalń i przedsiębiorstw z wymogami długofalowego programu rozwoju zastosowań informatyki w górnictwie węgla kamiennego. W wyniku tych prac wprowadzono komputeryzację do podstawowych dziedzin zarządzania, a tym samym stworzono efektywne systemy i metody gwarantujące wdrożenie modelu komputeryzacji zarządzania, opracowywanego na początku lat siedemdziesiątych w Głównym Instytucie Górnictwa.

Opracowanie i wdrożenie w COIG systemów i metod komputeryzacji w dziedzinie rozliczania i planowania działalności kopalń i przedsiębiorstw umożliwiło nie tylko udoskonalenie dotychczasowych metod i form rozliczeń oraz analizy działalności, ale również przyczyniło się do podniesienia ogólnego poziomu organizacyjnego tych jednostek. Dalsze prace w tej dziedzinie wiążą się ściśle z opracowywaniem całego kompleksu zagadnień dotyczących ekonomiczno-finansowych i organizacyjnych zasad funkcjonowania przedsiębiorstwa i zakładów przemysłu węglowego w warunkach reformy gospodarczej.

O obecnych rozmiarach działalności COIG, a zwłaszcza rozmiarach przetwarzania danych, świadczy najlepiej roczna liczba ponad 17 mln dokumentów wejściowych oraz kilkadziesiąt systemów i podsystemów informatycznych wykonywanych na zlecenie kilkudziesięciu użytkowników (łącznie ponad 2 tys. różnych dokumentów wynikowych).

SANOCKA FABRYKA AUTOBUSÓW

zatrudni natychmiast

informatyków z wyższym wykształceniem i kilkuletnią praktyką zawodową oraz dobrą znajomością komputera R-32 lub minikomputera MERA 305:

- do obsługi technicznej sprzętu komputerowego
- do projektowania i programowania systemów przetwarzania danych
- do programowania zastosowań inżynierskich (pożądana znajomość języka Fortran — JS).

Warunki pracy, płacy i mieszkaniowe do uzgodnienia w Dziale Kadr SFA, osobiście lub telefonicznie.
Adres: Sanocka Fabryka Autobusów, 38-500 Sanok, ul. Lipińskiego 109, tel.: 50-468

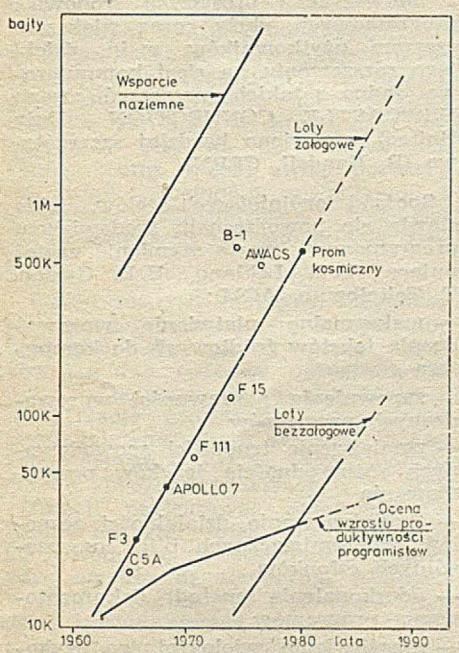
EO/133/K/86

Najnowsze tendencje w dziedzinie komputerowych systemów sterowania — problemy oprogramowania

Oprogramowanie jest jak entropia: trudno je objąć, nic nie waży i spełnia drugą zasadę termodynamiki. tzn. ciągle rośnie.
Bill Wulf, Tartan Laboratories

Tegoroczny przegląd [1]) prezentacji konferencyjnych w zakresie komputerowego sterowania ograniczymy do zagadnień programowania. Jest to celowe o tyle, że omówienie całej dziedziny przestało być możliwe w jednym numerze, tak bardzo się ona rozrosła, a ponadto, skupiając się na węższej tematyce, łatwiej przekazać więcej treści. Do przeglądu wybrałem konferencje, na których położono nacisk na metody wytwarzania oprogramowania:

- European Seminar on Industrial Software Engineering and the European Workshop on Industrial Computer Systems (EWICS), Freiburg, RFN, 12—18 kwietnia
- ACM Sigplan-Sigsoft Software Engineering Symposium on Practical Software Development Environments (ACM), Pittsburgh (stan Pensylwania), USA, 23—25 kwietnia
- European Physical Society Conference on Software Engineering Methods and Tools in Computational Physics (EUROPHYSICS), Bruksela, 21—24 sierpnia
- 1984 CERN School of Computing (CERN), Aiguablava, Hiszpania, 9—22 września
- IFIP WG 2.4 Working Conference on System Implementation Languages Experience and Assessment (IFIP), Canterbury, Wielka Brytania, 17—19 września.



Liczba bajtów kodu dla różnych systemów lotniczych i kosmicznych w czasie pierwszego lotu

W problematyce programowania na czoło wysuwa się zdecydowanie jedno zagadnienie: jak najlepiej i najtaniej wytwarzać oprogramowanie, co sprowadza się do odpowiedzi na pytanie, jak zwiększyć produktywność programistów? Dowodów na istnienie kryzysu w tej dziedzinie dostarczył B. Wulf (IFIP) przedstawiając wymowną ilustrację (rys.) szybkości wzrostu zapotrzebowania na oprogramowanie. Krzywa wzrostu produktywności programistów ma znacznie mniejsze nachylenie, co oznacza, że coraz trudniej jest sprostać potrzebom. W krajach rozwiniętych czyni się więc wszystko, aby zintensyfikować wytwarzanie oprogramowania.

JĘZYKI PROGRAMOWANIA

Jedną z metod zwiększania możliwości programistów jest wprowadzanie nowoczesnych języków wysokiego poziomu, zawierających konstrukcje uwalniające programistę od możliwie największej liczby szczegółów i zapewniające większą niezawodność oprogramowania. W tym kierunku modyfikuje się także starsze języki, np. FORTRAN (M. Metcalf, EUROPHYSICS). Obecnie opracowuje się nową normę FORTRANU, tzw. FORTRAN 8x. Choć prace nad tą normą jeszcze trwają i daleko do ich zakończenia, można wymienić najważniejsze proponowane zmiany:

— rezygnacja z formatowania programu źródłowego i dopuszczenie nazw o rozsądnej długości (do 31 znaków) oraz większości znaków ASCII, np. %, ?, ⟨i⟩ .
— typy i precyzja definiowalne przez użytkownika
— wprowadzenie agregatów i wycinków tablic
— dynamiczna gospodarka pamięcią przy użyciu instrukcji ALLOCATE i FREE
— modularyzacja programów przy użyciu instrukcji MODULE i USE
— rekursywność procedur i in.

Przewiduje się zachowanie pełnej zgodności z FORTRANEM 77, ale tylko przez ustalony okres. Przyczyną ciągłego doskonalenia FORTRANU jest wielkość dotychczasowych inwestycji w oprogramowanie napisane w tym języku. Dostrzegając przestarzałość wielu konstrukcji tego języka próbowano zresztą ulepszyć go znacznie wcześniej, tworząc RATFOR (B. QUIRK, EWICS), przedstawiony oddzielnie w tym numerze (str. 13).

Wymienione i inne nowo wprowadzone cechy FORTRANU upodobniają go w znacznym stopniu do takich języków, jak PL/I lub ADA i ewolucja ta jest nieunikniona. Wskazuje na to geneza ADY — od PASCALA przez MODULĘ (I. C. Pyle, EUROPHYSICS). Jedną z najważniejszych właściwości PASCALA było wprowadzenie typów definiowanych przez użytkownika. Główną konstrukcją MODULI jest moduł, będący rozszerzeniem struktury blokowej. W ADZIE rozwinięto te idee, łącząc je z mechanizmem rozłącznej kompilacji i wprowadzając obsługę wyjątków oraz współbieżność. Sądzi się, że ADA jest jednym z ostatnich języków imperatywnych (tj. takich, w których ciąg instrukcji wymusza jednoznaczne działania), które będą stopniowo wypierane przez języki funkcjonalne i deklaratywne (jak np. Prolog).

O ADZIE mówiono zresztą w kilku oddzielnych referatach nie dotyczących bezpośrednio samych konstrukcji języka, lecz ich użycia, poświęcając je projektowaniu i specyfikacji oprogramowania napisanego w ADZIE (G. Persch, IFIP), redukcji kosztów powtórnej kompilacji (M. Dausmann, IFIP), metodom obliczeniowym w zagadnieniach fizycznych (B. Wichmann, EUROPHYSICS) i samym kompilatorom (I. C. Pyle, EWICS). Cecha łącząca wszystkie zastosowania ADY jest niezbędność istnienia biblioteki matematycznej, obejmującej najważniejsze operacje obliczeniowe.

Języki, z których wywodzi się ADA, także ulegają znacznym rozszerzeniom. Przykładowo, z niezliczonej liczby rozszerzeń PASCALA omówiono mechanizmy obsługi wyjątków w języku MODULAR PASCAL (C. Bron, E. J. Dijkstra, IFIP) i sprzężenia międzymodułowe w PASCALU PLUS (D. W. Bustard, IFIP). Natomiast zapowiedzianych referatów na temat rozszerzeń MODULI-2 nie przedstawiono z powodu nieobecności autora (N. Wirth, EWICS, IFIP). Warto jednak wiedzieć, że dotyczą one m. in.[2]):

[1]) Poprzednie artykuły z tego cyklu opublikowano w numerach: 9/1980, 11—12/1981, 2/1983, 12/1983, 2/1985; Obecny przegląd stanowi rozszerzenie referatu przedstawionego w dniu 12 grudnia 1984 r. na seminarium Sekcji Języka Ada Polskiego Towarzystwa Informatycznego

[2]) N. Wirth: Revisions and Ammendments to Modula-2. Journal of Pascal Ada and Modula-2, Vol. 4, No. 1, pp. 25—28 (1985)

— eksportowania wszystkich obiektów modułu definicyjnego

— składni rekordów wariantowych i instrukcji case

— parametrów procedury TRANSFER

— składni typu zbiorowego, zawężonego i typu ADDRESS i in.

Tłumaczenie odpowiedniego raportu ukaże się niebawem w INFORMATYCE.

Ciekawe rozszerzenia wprowadzono także do języka C, nazywając go C++ (B. Stroustrup, IFIP) [3, 4] oraz do CHILLA (K. Rekdal, IFIP) [5, 6]. Język C uzupełniono takimi konstrukcjami, jak klasy, przeciążenie operatorów i funkcji, odsyłacze i in., a do CHILLA dodano rozłączną kompilację, obsługę wejścia-wyjścia i dynamiczną alokację pamięci.

Spośród całkowicie nowych języków wyróżniają się: OCCAM (D. May, IFIP) i TURING (J. Cordy, IFIP), będący następcą dość znanego języka EUCLID [7]. OCCAM jest językiem całkowicie nowym [8], opracowanym specjalnie dla wieloprocesorowego mikrokomputera o nazwie TRANSPUTER [9]. Podstawową cecha języka jest równoległość operacji i łatwość programowania komunikacji wieloprocesorowej.

Na wymienionych konferencjach omawiano jeszcze wiele innych języków, lecz sądzę, że ze względu na ich małe rozpowszechnienie można je w tym przeglądzie pominąć. Interesujące jest natomiast zalecenie opublikowane przez grupę studialną języków programowania projektu ESPRIT [10], w którym sformułowano uzasadnienie dotyczące wyboru zaledwie pięciu języków (PASCAL, C, ADA, LISP i PROLOG) do stosowania w nowych przedsięwzięciach programistycznych oraz dopuszczenie FORTRANU 77 do stosowania w pracach wymagających użycia istniejącego oprogramowania.

ŚRODOWISKA PROGRAMOWE

Obecnie panuje już duża zgodność co do tego, że choć nowoczesne języki wysokiego poziomu i ich kompilatory mają podstawowe znaczenie dla zwiększenia zdolności do wytwarzania oprogramowania, to ich istnienie rozwiązuje zaledwie niewielką część całego problemu. Do efektywnego stosowania języków konieczne są różnorodne narzędzia programowe, które zintegrowane wokół określonego języka, modelu czy metodologii tworzą środowiska programowe (ang. programming support environments).

Zagadnieniom tym poświęcono oddzielną konferencję (ACM), gdzie spotkali się wszyscy czołowi twórcy najbardziej znanych przedsięwzięć tego rodzaju, jak GANDALF [11] (A. N. Habermann, D. Notkin) czy MENTOR [12] (V. Donzeau-Gouge i in.), a na pozostałych konferencjach zajęły one również wiele miejsca. Dość charakterystyczne jest, że środowiska programowe tworzy się często z już istniejących narzędzi (bądź w ramach istniejących struktur), dobudowując nowe, bardziej złożone i nowoczesne.

Tak powstało, na przykład, środowisko programowe DAISY (G. Bull, EWICS) przeznaczone do pisania oprogramowania w BASICU dla systemów rozłożonych przestrzennie. Jego podstawowymi elementami są: opis konfiguracji, opis otoczenia i opis powiązań wzajemnych. Posiadając te informacje przechowywane w relacyjnej bazie danych, tworzy się system docelowy przy użyciu automatycznych narzędzi i zastosowaniu metod graficznych. Sprzężenia narzędzi (ang. tool interface) są wydzielone z bazy danych, a sprzężenie do współpracy z użytkownikiem (ang. man-machine interface) jest oddzielone od narzędzi.

Jedno ze środowisk PASCALA, PERSPECTIVE (K. Jackson, EWICS), ma bardziej formalne podstawy. Oparte jest na opracowanym wcześniej systemie MASCOT (ang. Modular Approach to Software Construction Operation and Test) i zestaw narzędzi CONTEXT, stanowiący realizację tego podejścia dla języka CORAL-66. Niektóre idee zaczerpnięto ze środowiska programowego ADY — APSE (ang. Ada Programming Support Environment). Według autorów, w realizacji oprogramowania w tym środowisku można stosować różne formalizmy, tj. strukturalnie różne modele systemów, różne metodologie, tj. podejścia organizacyjne do projektowania, kodowania, testowania i pielęgnacji oprogramowania oraz — różne zbiory narzędzi. Podobną budowę ma inne środowisko, opracowane w BFN, znane pod nazwą S/E/TEC (W. Hesse, EWICS), stosowane głównie do wytwarzania oprogramowania dla systemów mikrokomputerowych.

Największe nakłady przeznacza się obecnie na rozwój środowisk programowych ADY (K. Ripken, EWICS). Założono, że powinny one zawierać standardowe jądro, tzw. KAPSE (ang. Kernel APSE), złożone z bazy danych i systemu wykonawczego (ang. run-time system), zapewniające niezależne sprzętowo sprzężenie z resztą APSE. Otoczeniem jądra jest tzw. środowisko minimalne, składające się z minimalnego zestawu narzędzi napisanych w ADZIE, koniecznych i wystarczających do opracowywania programów. Pełne środowisko APSE stanowi rozszerzenie wersji minimalnej w kierunku określonych zastosowań lub mtodologii programowania. Do chwili obecnej powstało kilka takich środowisk, przede wszystkim w USA — ALS (ang. Ada Language System), AIE (ang. Ada Integrated Environment), ADE (ang. Ada Development Environment) i ARCTURUS.

Jest interesujące, że w jednym z nich, AIE, skorzystano w istotny sposób z koncepcji UNIXA (J. Favaro, EWICS), poddając je jednak istotnym uzupełnieniom. Przykładowo, zamiast zwykłego hierarchicznego systemu plików wprowadzono złożoną bazę danych, służącą do przechowywania obiektów. Podobnie, jednolite jądro i interpreter poleceń zastąpiono bardziej złożonymi strukturami warstwowymi, zbudowanymi — odpowiednio — z pakietów i procedur. UNIX będący znakomitym przykładem jednego z pierwszych i prostszych środowisk programowych, służy coraz większym rzeszom użytkowników, m. in. dzięki jego rozszerzeniu na sieci komputerowe. Jedno z takich rozszerzeń, zwane NEWCASTLE CONNECTION (Unix United), omówiono bardziej szczegółowo (B. Randell, CERN).

Spośród ogólniejszych celów, jakie stawia się nowoczesnym środowiskom programowym, warto wymienić następujące (D. B. Leblang, B. B. Garlan, T. Teitelbaum, ACM):

— maksymalne ułatwienie wprowadzania tekstów źródłowych do komputera

— zapewnienie poprawności wprowadzanych programów

— zmniejszenie trudności spowodowanych różnorodnością języków programowania

— skoordynowanie działalności zespołów programistycznych i ich poszczególnych członków

— udoskonalenie metod dokumentowania programów

— umożliwienie wielokrotnego wykorzystania dużych segmentów oprogramowania itp.

Do realizacji tych celów stosuje się różnorodne środki (A. I. Wasserman, EUROPHYSICS), do których należą nie tylko narzędzia programistyczne, obejmujące oprócz tradycyjnych edytorów, kompilatorów, programów łączących, ładujących i testujących znaczną liczbę bardziej specyficznych programów, jak analizatory statyczne

---

[3]) J. Winiewski: Język programowania C. INFORMATYKA, nr 4—6 (1985)

[4]) B. Stroustrup: A C++ Tutorial. Report CSTR-113. The C++ Programming Language Reference Manual. Report CSTR-108. Bell Laboratories, Murray Hill (NJ), 1984

[5]) W. Udrycki, W. Wilczyński: CHILL — język programowania systemów komutacyjnych. INFORMATYKA, nr 9—11 (1984)

[6]) CCITT Recommendation Z. 200: Chill Language Definition. Chill Bulletin, Vol. 5, No. 1, January 1985

[7]) J. R. Cordy, R. C. Holt: Specification of Concurrent Euclid. Report CSGR-133, University of Torento, 1981

[8]) D. May: Occam. SIGPLAN Notices, Vol. 18, No. 4, pp. 69—79 (1983)

[9]) P. Eckelmann: Architektur und Anwendung des Transputers. Elektronik, Nr 4, ss. 59—65, 24 Februar 1984

[10]) European Strategic Programme for Research in Information Technology

[11]) A. N. Habermann: The Gandalf Research Project. Carnegie-Mellon University, Pittsburgh (PA), 1980

[12]) V. Donzeau-Gouge et al.: Programming Environments Based on Structured Editors — The Mentor Experience. Report No. 26, INRIA, July 1980

i dynamiczne, administratory plików, zarządcy konfiguracji i in., lecz także:
— wspólne bazy danych i protokoły komunikacyjne służące do zintegrowania narzędzi
— udoskonalone mechanizmy dokumentacyjne
— ścislejsze powiązania narzędzi z konkretną metodologią
— doskonalszy sprzęt, głównie indywidualne stanowiska robocze (ang. workstations), powiązane wzajemnie w sieci komputerowe, zamiast dotychczasowych systemów z podziałem czasu
— lepsze, tzn. bardziej „przyjazne" (ang. user-friendly) sprzężenia użytkowe, korzystające z doskonalszych monitorów graficznych, techniki okienkowania oraz myszy
— lepsze warunki ergonomiczne.

ZAGADNIENIA SPECYFIKACJI

Tak rozumiane środowiska programowe stanowią na ogół element składowy całych metodologii programowania (J. K. Buckle, EUROPHYSICS), służących do wspomagania procesów programistycznych w całym okresie istnienia oprogramowania. Dotąd wypracowano wiele metodologii [13]), z których najbardziej znane i skuteczne są podejścia formalne (HDM), graficzne (SADT), analityczne (PSL-PSA) i strukturalne (MASCOT). Jedna z nowszych metod, JSD (ang. Jackson System Development) polega na tworzeniu opisu systemu przy użyciu jego działań, ich atrybutów i zależności czasowych między nimi (J. R. Cameron, EUROPHYSICS).

W całym cyklu produkcyjnym oprogramowania za najważniejszą uważa się fazę specyfikacji wymagań, gdyż jak stwierdzono: im wcześniej wykryje się błąd w programie, tym łatwiejszy jest on do poprawienia. Dlatego coraz więcej uwagi poświęca się samym metodom specyfikacji i technikom transformacji wymagań specyfikacyjnych na gotowy kod programu.

Wśród różnych rodzajów specyfikacji na ogół odróżnia się specyfikację wymagań, sporządzoną przez użytkownika lub odbiorcę oprogramowania i — powstałą na tej podstawie — specyfikację projektową, sporządzoną przez implementatora (wykonawcę) oprogramowania. Już na etapie przejścia od specyfikacji wymagań do specyfikacji projektowej podejmuje się istotne decyzje projektowe, które mogą powodować powstawanie defektów, gdyż zwykła dekompozycja ogólnych wymagań werbalnych (celów) na bardziej sformalizowane zadania szczegółowe prowadzi na ogół do nadmiernych uproszczeń (E. Knuth, J. Ludewig, EWICS). Jeżeli tę wątpliwość rozszerzy się na kolejne etapy uszczegółowiania projektu, to okaże się, że stosowana powszechnie metoda kolejnych uściśleń (projektowanie zstępujące) jest bardzo ryzykowna, gdyż nie uwzględnia zmian jakościowych przy przejściu na poziom bardziej szczegółowy (a tylko zmiany ilościowe).

Jedną z ważniejszych sformalizowanych metod opracowywania oprogramowania (A. Beer, EWICS) jest tzw. HDM (ang. Hierarchical Development Methodology), dość szeroko opisywana w literaturze [14]). Najważniejsze etapy tworzenia specyfikacji przy użyciu tej metody można ująć w kilku punktach:
— tworzenie koncepcji
— zdefiniowanie maszyny abstrakcyjnej i sprzężeń
— specyfikacja modułów w języku SPECIAL [15])
— przedstawienie danych w tym języku.

Zaimplementowania tak otrzymanej specyfikacji można dokonać w dowolnym języku programowania, tak więc sama metoda jest dość uniwersalna. Tą metodą można również otrzymać specyfikację interakcyjnie przy użyciu konwencjonalnych komputerów i zwykłych systemów operacyjnych. Jej inną zaletą jest duża liczba wbudowanych narzędzi, szczególnie do testowania i weryfikacji. Zarówno sama metoda HDM, jak i język SPECIAL, mają pewne wady, np. niewyrażalność właściwości dynamicznych (HDM) czy nieczytelność (SPECIAL), lecz są zalecane do projektowania dużych systemów operacyjnych i baz danych. Metoda HDM, podobnie jak inne systemy tego rodzaju, ewoluuje w kierunku objęcia swoim zasięgiem całego okresu istnienia oprogramowania i — utworzenia kompletnego środowiska programistycznego.

System SARS (ang. System for Application-oriented Requirements Specification) jest przeznaczony głównie do stosowania przy produkcji oprogramowania przemysłowego (w sterowaniu procesami). Jest on oparty na określonej metodzie ułatwiającej specyfikowanie wymagań użytkowych i — narzędziach dostosowanych do wykonywania odpowiednich czynności. W przyjętej metodzie korzysta się z naturalnego faktu, że oddziaływanie środowiska na sterowany obiekt polega na nadzorowaniu jego zachowania i zastosowaniu odpowiednich akcji według przyjętego algorytmu postępowania. Etapy odpowiadające tym czynnościom stanowią trzy fazy postępowania zmierzającego do zanalizowania wymagań. Właściwe sformułowanie wymagań następuje w języku LARS (ang. Language for ARS), odbywa się przy użyciu automatycznych narzędzi i polega na transformacji wymagań nieformalnych. Głównymi składnikami języka są konstrukcje odpowiadające trzem wymienionym rodzajom wymagań, tzn.:
— szczegółowy opis sprzężeń procesu sterowanego z otoczeniem
— lista przewidywanych zadań systemu sterowania
— zależności funkcjonalne między bodźcami i odpowiedziami systemu sterowania.

Według relacji autora (W. K. Epple, EWICS) wymieniony język zapewnia gładkie przejście od opisu werbalnego systemu do specyfikacji formalnej, dzięki czemu możliwe jest zautomatyzowanie procesu wytwarzania oprogramowania.

Różne aspekty specyfikacji wymagań dla przemysłowych systemów bezpieczeństwa przesądzają o daleko większej złożoności tego problemu (W. Quirk, EWICS). W dokumencie Komitetu TC7 EWICS (Safety and Security) dotyczącym tej problematyki stwierdza się, że bezpieczeństwo systemu nie jest jego cechą wewnętrzną lecz zależy od interakcji z otoczeniem, tak więc kluczowym elementem specyfikacji jest określenie odpowiednich zależności środowiskowych. Ponieważ właściwości otoczenia zmieniają się w czasie, specyfikacja wymagań powinna dotyczyć całego cyklu istnienia systemu, od fazy opracowywania, przez fazy produkcji i eksploatacji, aż do jego wycofania z użycia. W dokumencie tym wyróżniono cztery podsystemy, których wymagania składają się na pełną specyfikację dostarczanego systemu:
— system docelowy
— środowisko systemu docelowego
— projekt wytwarzania systemu docelowego
— środowisko projektowe.

* * *

W przedstawionym omówieniu nie uwzględniono wielu innych zagadnień programowania komputerowych systemów sterowania, prezentowanych podczas wymienionych konferencji, jak: bazy danych czasu rzeczywistego (P. Jervis, EWICS), współpraca człowiek-maszyna (H. G. Stassen, EWICS) czy systemy graficzne (F. R. A. Hopgood, EUROPHYSICS). Dokonany podział tematyki też jest dość sztuczny, gdyż trudno jest arbitralnie rozdzielić integralnie związaną problematykę języków programowania, środowisk programistycznych i metodologii programowania — dokonano go jedynie w celu zwiększenia przejrzystości materiału. Ucierpiała na tym szczególnie ta ostatnia dziedzina, którą omówiono zbyt powierzchownie, ograniczając się do zagadnień specyfikacji, a pomijając pozostałe fazy cyklu istnienia oprogramowania, m. in. niezwykle ważną problematykę kontroli konfiguracji w procesie opracowywania systemów (C. Mazza, EUROPHYSICS).

Pominięto też w całości prezentowane zastosowania, a pewne zdziwienie może budzić brak bliższego omówienia systemów operacyjnych — nie jest to jednak przypadek. Systemy operacyjne traktuje się obecnie jako część maszyny wirtualnej i rzadko są one elementem dostawy oprogramowania użytkowego.

[13]) R. Żelazny: Narzędzia inżynierii oprogramowania. INFORMATYKA nr 10—11, 1984

[14]) B. A. Silverberg: An Overview of the SRI Hierarchical Development Methodology. H. Hünke, ed., Software Engineering Environments. North-Holland, Amsterdam, 1981

[15]) C. S. Chandersekaran, R. C. Linge: Software Specification Using the SPECIAL Language. Journal of Systems and Software, Vol. 2, No. 1, pp. 31—38 (1981)

Ogromne zainteresowanie towarzyszy obecnie rozwojowi systemów ekspertowych (F. Varedejo, CERN; R. Yasdi, EUROPHYSICS) systemom baz wiedzy (G. H. F. Diercksen, R. Studer, EUROPHYSICS), superkomputerom (T. Bloch, CERN; S. Kamiya, EUROPHYSICS) i komputerom tzw. piątej generacji (P. C. Treleaven, EUROPHYSICS). Jest to jednak problematyka tak odległa od krajowych zastosowań komputerów, że z tego powodu również nie uwzględniono jej w przeglądzie.

Jednakże Czytelnicy bliżej zainteresowani tematyką, którą pominięto w tym omówieniu, jak również pragnący uzyskać bardziej szczegółowe informacje na którykolwiek z omawianych tematów, mogą sięgnąć do opublikowanych w całości materiałów konferencyjnych:

- EWICS — V. H. Haase, R. Hammer, eds., Softwaretechnologie für die Industrie, R. Oldenburg, Wien, 1984
- ACM — SIGPLAN Notices, Vol. 19, No. 5, May 1984
- EUROPHYSICS — Computer Physics Communications, Vol. 38, No. 1, 1985
- CERN — Proc. 1984 School of Computing. Report 85—09, Geneva, 1985
- IEIP WG 2.4 — P. King. ed., System Implementation Languages, North-Holland, Amsterdam, 1985.

JANUSZ ZALEWSKI

Terminologia

DIXIT ALGORIZMI
czyli o pochodzeniu i znaczeniach słowa algorytm

Pochodzenie słowa algorytm sięga czasów wczesnego średniowiecza. Wówczas to, za kalifatu Omajadów, Arabowie zawładnęli częścią Indii, w końcu VII w. opanowali Chorezm i Bucharę, a w 719 r. przeprawili się przez Pireneje. W obrębie tych ogromnych posiadłości znalazły się kraje o rozwiniętej kulturze. Arabowie stanowili ten nieliczny rodzaj zwycięzców, którzy nie tylko nie tępili zdobyczy cywilizacyjnych podbitych narodów, ale przyswajali je sobie i rozpowszechniali.

Około 400 r. Hindusi wynaleźli dziesiętny system liczbowy — 9 cyfr i puste miejsce, później oznaczane kropką lub kółkiem. Jednym z pierwszych entuzjastów rachunku indyjskiego był Muhammad ibn Musa al-Chorezmi (ok. 787 — ok. 847). Przydomek al-Chorezmi wskazuje jego ojczyznę — był nim Chorezm, kraj w dolnym biegu Amu-darii.

W księdze „O rachunku indyjskim" al-Chorezmi przedstawił korzyści wynikające ze stosowania tego rachunku, podał wiadomości o cyfrach indyjskich (dziś zwanych arabskimi), objaśnił zasadę dziesiętnego pozycyjnego systemu reprezentacji liczb, zademonstrował metody dodawania, odejmowania, mnożenia i dzielenia liczb całkowitych oraz mnożenia i dzielenia ułamków. Dzieło jest znane wyłącznie z niekompletnego tłumaczenia na język łaciński, zwanego „De numero indorum". Oryginał arabski napisany ok. 825 r. dotąd nie został odnaleziony [9]. Wymienione tłumaczenie, znalezione w jednym z klasztorów angielskich, pochodzi sprzed roku 1143. Obecnie znajduje się w zbiorach biblioteki uniwersytetu w Cambridge. Początkowy jego fragment brzmi [1]: *„Powiedział Algorizmi: Wznieśmy należytą chwałę Bogu, władcy i obrońcy naszemu, okażmy Mu należne względy i chwałę pomnóżmy uwielbieniem, módlmy się do Niego, żeby skierował nas na ścieżkę sprawiedliwości, żeby prowadził nas drogą prawdy, żeby pomógł nam wypełnić to, co zdecydowaliśmy wyjaśnić o rachunku indyjskim z pomocą .IX. liter, którymi oni wyrażali dowolną swą liczbę dla łatwości i zwięzłości, ułatwiając pracę temu, kto studiuje arytmetykę, tj. liczbę największą i najmniejszą, i wszystko co jest w niej z mnożenia i dzielenia, dodawania, odejmowania i tak dalej.*

Powiedział Algorizmi: Kiedy zobaczyłem, że Hindusi zestawiali z .IX. znaków każdą swą liczbę, dzięki położeniu jakie ustanowili, zapragnąłem wyjawić, o ile spodoba się to Bogu, co otrzymuje się z tych liter, by ułatwić uczącemu się".[1])

Brakujące fragmenty dzieła al-Chorezmiego próbowano zrekonstruować posługując się pokrewnymi pracami z XII i XIII w. Najstarsza z nich pochodzi z roku 1143 i znajduje się w Wiedniu. Inna, sprzed 1180 r., przechowywana w Paryżu, rozpoczyna się słowami: „Incipit prologus in libro Alghoarismi de pratica arismetice. Qui editus est magistro Johanne Yspalensi" (Rozpoczyna się prolog do księgi Alghoarismi o praktyce arytmetycznej. Wydanej przez magistra Jana Hiszpańskiego).

Od XIII w. następuje w Europie rozkwit twórczości matematycznej. Pierwszym znakomitym uczonym tego okresu był Leonardo z Pizy (ok. 1170 — ok. 1250) znany jako Fibonacci, autor „Liber abaci". Szerokie oddziaływanie miały także dwa inne dzieła: Jana de Sacrobosco (I poł. XIII w.) „Tractatus de arte numerorum" i Aleksandra de Villedieu (1170—1250) „Carmen de algorismo" (Pieśń o algorytmie) [15].

Powiedział Algorizmi (dixit Algorizmi) — zlatynizowana postać nazwiska al-Chorezmi dała początek słowu *algorytm*. W średniowiecznej Europie oznaczało ono *system dziesiętnej arytmetyki pozycyjnej*.

Nie było to jedyne znaczenie tego słowa. W języku francuskim *algorytm* (algorisme) oznaczał także *chronogram, czyli taki układ wyrazów łacińskich lub wierszy, w którym pewne (tj. pierwsze lub wyróżnione graficznie) litery składały się na liczbę rzymską oznaczającą datę wydania książki lub datę wydarzenia*. Ponadto, wyraz stosowany był w zwrocie idiomatycznym *être chiffre en algorisme* (być cyfrą w algorytmie), który oznaczał *być niczym, być zerem, nie mieć żadnej wartości* [6].

Z czasem zaczęto zapominać o pochodzeniu słowa algorytm. Etymologię jego wywodzono z greki łącząc ją na przykład ze słowem arithmos (liczba). Wiązano ją także z ludźmi, na przykład z Algusem, rzekomo znamienitym filozofem [15]. Jeszcze w XIX w. niektórzy sądzili, że pochodzi od arabskiego al i greckiego arithmos, a pojawienie się w wyrazie litery g tłumaczono pomyłką, która niby miała nastąpić przy przepisywaniu rękopisów łacińskich. Przeciwstawił się tym wywodom w r. 1845 Reinaud [16]. Nie była to zresztą jedyna praca przypominająca al-Chorezmiego. W 1857 r. B. Boncompagni wydał w Rzymie „Trattati d'Aritmetica publicati da Baldassare Boncompagni. I. Algoritmi de numero indorum. II. Joanni Hispalensis liber algorismi de pratica arismetice", w której zawarł tekst rękopisu z Cambridge i pracę wydaną przez Jana Hiszpańskiego. Jak należy rozumieć słowo algorytm, wywodził też w swej pracy Hunrath [8], cytując XVI-wieczne dzieło, w którym mówi się, że *algorytmem* nazywa się *sztukę liczenia*.

Przejścia od *sztuki liczenia* do *sposobu postępowania przy prowadzeniu obliczeń* należy szukać u G. W. Leibniza

[1]) Średniowieczny tłumacz używa w przekładzie stosowanych wówczas powszechnie w Europie liczb rzymskich

(1646—1716) [11]. W opublikowanej w 1684 r. w czasopiśmie Acta Eruditorum pracy pt. „Nowa metoda maksimów i minimów, a także stycznych, której nie opierają się wielkości ani ułamkowe, ani niewymierne, tudzież szczególny dla nich rodzaj rachunku", dotyczącej wynalezionego przez siebie rachunku różniczkowego, pisał: *„Jeśli się zna tzw. Algorytm tego rachunku, który nazywamy różniczkowym, to wszystkie pozostałe równania różniczkowe otrzymać można za pomocą ogólnej metody rachunkowej, i można będzie znajdować maksima i minima, a także styczne, nie narażając się przy tym na konieczność usuwania ułamków, czy niewymierności, czy innych skomplikowanych wyrażeń, co jednak trzeba było czynić posługując się metodami dotychczas ogłoszonymi"* [10].

Znaczenie, jakie słowo algorytm ma we współczesnej teorii obliczalności, zawdzięczamy m. in. pracom A. M. Turinga (1912—1954), S. C. Kleene'a (ur. 1909), A. Churcha (ur. 1903), E. L. Posta (1897—1954), A. A. Markowa (ur. 1903). W latach trzydziestych opracowano teorie prowadzące do matematycznego udokładnienia nieprecyzyjnego pojęcia algorytmu. Dotąd *algorytm* oznaczał *dokładny przepis określający konstruktywny proces obliczeń, prowadzący od danych początkowych do wyniku.* Zakładano, że przepis ten nie powinien dopuszczać nieprzewidzianych dowolności. W każdym kroku procesu jednoznacznie określa się, czy należy wykonać następny krok, a jeśli tak, to na czym powinien on polegać. Wynikiem każdego kroku jest pewien konstruktywny obiekt. Od tego, jaki jest wynik, zależy następny krok procesu lub jego brak. Dane początkowe wpływające na pierwszy krok, mogą zmieniać się w pewnych zakresach. Takie określenie algorytmu nie stanowi jego ścisłej definicji matematycznej; pojęcie konstruktywności procesu czy obiektu jest mgliste [14].

Teorie opracowane przez Kleene'a, Churcha i Turinga, wszystkie opublikowane w 1936 r., korzystają z pojęcia funkcji częściowo rekurencyjnej. W każdej — choć różnią się od siebie co do zakresu i szczegółów — przedstawiono formalną charakterystykę obszernej klasy funkcji częściowych. Wszystkie prace „miały wspólne cechy (...), po pierwsze, dając (...) formalne odpowiedniki pojęcia *algorytmu* (dla funkcji częściowych), a po drugie, w konsekwencji (...) dając odpowiedniki pojęcia *funkcji częściowej obliczalnej przez algorytm"* [17]. Wyniki Posta — opublikowane dużo później, choć wyprzedzały prace Turinga [4] — dotyczyły także analizy procesu obliczeniowego. W r. 1951 Markow [13] zaproponował uściślenie pojęcia algorytmu za pomocą wprowadzonego przez siebie pojęcia algorytmu normalnego [2]).

Wszystkie te prace, a także inne, o których tu nie wspomniano, doprowadziły do wykazania, że klasy funkcji częściowych obliczalnych przez algorytm definiowane za pomocą różnych formalizmów stanowią tę samą klasę funkcji częściowych.

Przedstawiwszy ewolucję znaczenia wyrazu algorytm w literaturze światowej, nie można zapominać o jego rozumieniu w dziełach autorów polskich. Biskup krakowski Wincenty Kadłubek (ok. 1150—1223) opisując w swej łacińskiej kronice bitwę Krzywoustego pod Nakłem zanotował: „Liczbę poległych poznać byś mógł chyba policzywszy piasek w morzu lub gwiazdy na niebie. A jeśli masz czas po temu, poradź się algorystów, którzy są biegli w rachowaniu (...)" [5].

Z XV-wiecznych manuskryptów matematycznych wymieńmy trzy świadczące także o tym, że wówczas algorytm był synonimem arytmetyki [5].

- Mikołaja z Polski „Algorismus prosaicus Johannis de Sacrobosco";
- Marcina Króla (ok. 1422—1453) „Algorismus minutiarum" (Rachunek ułamków);
- nieznanego autora „Nova compilatio Algorismi minutiarum" (Nowe zestawienie rachunku ułamków).

W dobie odrodzenia zaczynają ukazywać się książki w języku polskim. Dwie pierwsze z dziedziny matematyki, to [2, 5]:

- z r. 1538 „Algoritmus: to jest nauka Liczby: Polską rzeczą wydana: Przez Księdza Tomasza Kłosa. Na trzy się części dzieli. Pierwsza będzie o osobach Liczby, Wtora o Regule detri, Trzecia o rozmaitych rachunkach y o społkach kupieczkich" (fot.),

Algoritmus: To
iesth nauka Liczby: Polską
rzeczą wydana: Przez Xiędza Tomasza
Kłosa. Na trzy sie części dzieli. Pierwsza bę-
dzie o osobach Liczby, Wtora o Regule
detri, Trzecia o rozmaitych rachun-
kach y o społkach Kupieczkich.
C Krakowie w Officinie Unglerianie 1538.

„Algoritmus" T. Kłosa

- wydany w r. 1553 przez Bernarda Wojewódkę „Algorithm, to jest nauka liczby po polsku na liniach teras nowo z pilnością przeyrzany, na wielu miejscach poprawiony, y snadnie k nauczaniu nisz pierwey podany".

Marian Baraniecki (1848—1895), profesor Uniwersytetu Jagiellońskiego, komentując „Algoritmus" ks. Kłosa pisał [2]: „Długo najdziwniej objaśniano ten wyraz (algorithmus, algorismus to późniejsze wersyje; wcześniejsze są: algorithmi, algorismi, alchaurismi itd.), oznaczający arytmetykę cyfrową; tak np. w pierwszym w Polsce drukowanym podręczniku arytmetycznym łacińskim de Sacro Busto (1509) czytamy we wstępie: enim nomen ipsus Algorithmus et decitur ab algos, id est ars, et rithmus, quod est numerus, quasi ars numerandi [3]). Dopiero kiedy znaleziony w bibliotece w Cambridge łaciński rękopis (...) okazał się tłumaczeniem arytmetyki (cyfrowej) arabskiej (...), wyjaśniło się, że wyraz algoritmi i jego odmiany jest przeróbką wyrazu Alchwarizmi (...). Kłos w tytule swojej książki (podobnie jak np. wcześniejszy Jan z Łańcuta i późniejszy Wojewódka) przez algoritmus rozumie w ogóle arytmetykę. Obecnie w matematyce używa się wyrazu algorytmu na oznaczenie powtarzającego się postępowania w rachunku, które staje się prawidłem".

Wiek XIX zaowocował pracami Józefa Marii Hoene-Wrońskiego (1776—1853), który w swych dziełach wyrażał głębokie przekonanie o zasadniczej roli algorytmu w matematyce i filozofii. W r. 1810 przedstawił Instytutowi Francuskiemu rozprawę pt. „Premier principe des methodes des algorithmiques" (Główna zasada metod algorytmicznych). Rozwinięcia jej dokonał w „Introduction a la philosophie des mathématiques, et téchnie de l'algorithmie" (Wstęp do filozofii matematyki oraz technia algorytmii) Wroński wyróżniał dwa podejścia do badań matematycznych. Pierwsze, to technika algorytmiczna, polegająca na tworzeniu środków przeznaczonych do wyznaczania wartości wielkości matematycznych, drugie, to teoria z celem w postaci badania istoty pojęć matematycznych. Według niego „technia algorytmii ma za przedmiot *miarę* czyli *obliczanie* ilości algorytmicznych, zaś *teoria* algorytmii ma za przedmiot *naturę* samą czyli konstrukcję tych ilości" [7].

[2]) Z uwagi na to, że rubryka terminologiczna dotyczy jedynie rozważań językowych, nie podajemy formalnych definicji algorytmu; można je znaleźć w cytowanych publikacjach (przyp. red.)

[3]) W wolnym przekładzie: przecież jego właśnie imię Algorithmus, a było wywiedzione od algos, to jest sztuka, i rithmus, który oznacza liczbę — jakoby sztuka rachowania

W przeciwieństwie do prac matematycznych i filozoficznych słowniki i encyklopedie podawały potoczne rozumienie algorytmu.

W „Słowniku języka polskiego" Samuela Lindego z r. 1807 brak jest hasła algorytm. Znajduje się ono natomiast w encyklopedii Orgelbranda [19], gdzie przez algorytm rozumie się *„zbiór wszystkich znaków, używanych w jakimkolwiek rachunku"*. Pochodzenie wyrazu wywiedzione jest od „nazwiska matematyka arabskiego Alkarezmi".

W „Słowniku języka polskiego" z roku 1900 [18] wyróżnia się dwa znaczenia tego słowa. Jest to:

- „system symbolicznego oznaczania form i działań logicznych; symboliczne wyrażanie działań nad pojęciami; system znaków lub symbolika logiczna";
- „rachunek przy pomocy cyfr lub metod arabskich" bądź „pewna ściśle określona metoda albo schemat postępowania rachunkowego ⟨od nazwiska matematyka arabskiego Al-Chowaresmi (= Chowarezmijczyk)⟩".

W międzywojennej encyklopedii Gutenberga [21] „algorytm oznacza działanie matematyczne. Pochodzi od arabskiego imienia Alchwarizmi".

JANUSZ STOKŁOSA

LITERATURA

[1] Al-Chorezmi Muhammad ibn Musa: Matematičeskie traktaty. FAN, Taškent, 1983
[2] Baraniecki M. A.: Algoritmus, to jest nauka liczby, polską rzeczą wydana przez księdza Tomasza Kłosa — 1538. Wydawnictwa Akademii Umiejętności, Kraków, 1889
[3] Church A.: An unsolvable problem of elementary number theory. American Journal of Mathematics, Vol. 58, 1936, 345—363
[4] Davis M.: Czym jest obliczenie? Steen L. A. (red.) Matematyka współczesna. Dwanaście esejów. WNT, Warszawa, 1983
[5] Dianni J., Wachułka A.: Tysiąc lat polskiej myśli matematycznej. PZWS, Warszawa, 1963
[6] Godfroy F.: Dictionnaire de l'Ancienne Langue Française et de tous ses Dialectes du IXe au XVe siècle. Paris, 1880
[7] Hoene-Wroński J. M.: Filozofia matematyki. Instytut Wydawniczy „Biblioteka Polska", Warszawa, 1937
[8] Hunrath K.: Zum Verständnis des Wortes Algorismus. Bibliotheca mathematica, F. 2, Bd. 1, 1887
[9] Juškevič A. P.: O trude po arifmetike Muhammada ibn Musy al-Chorezmi, [1], 150—202.
[10] Juszkiewicz A. P. (red.): Historia matematyki. PWN, Warszawa, 1975—1977
[11] Kaufmann H.: Dzieje komputerów. PWN, Warszawa, 1980
[12] Kleene S. C.: General recursive functions of natural numbers. Mathematische Annalen, Vol. 112, 1936, 727—742
[13] Markov A. A.: Teorija algorifmov. Trudy Matematičeskogo Instituta AN SSSR, Tom 38, 176—189
[14] Markov A. A.: Elementy matematičeskoj logiki. Izd. Moskovskogo Universiteta, Moskva, 1984
[15] Matvievskaja G. P.: Vydajuščijsja matematik Muhammar ibn Musa al-Chorezmi i literatura o nem, [1], 203—300
[16] Reinaud J. T.: Mémoire géographique, historique et scientifique sur l'Inde antérieurement au milieu du XIe siècle de l'ère chrétienne. Mémoire de l'Académie des Inscriptions et belles Lettres de France, t. XVIII, 2, 1845
[17] Rogers H. Jr.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York, 1967
[18] Słownik języka polskiego ułożony pod redakcją Jana Karłowicza, Adama Kryńskiego i Władysława Niedźwieckiego, t. 1, Warszawa, 1900
[19] S. Orgelbranda Encyklopedja Powszechna, t. 1, Warszawa, 1898
[20] Turing A. M.: On computable numbers, with an application to the Entscheidungsproblem. Preceedings of the London Mathematical Society, Ser. 2, Vol. 42, 1936, 230—265; Vol. 43, 1936, 544—546
[21] Wielka Ilustrowana Encyklopedja Powszechna Wydawnictwa „Gutenberg", Kraków.

Recenzje

Systemy informacyjne: aspekty teoretyczne i formalne

W kwietniu 1985 r. znana oficyna wydawnicza North-Holland w swej serii IFIP wydała pozycję zatytułowaną „INFORMATION SYSTEMS: THEORETICAL AND FORMAL ASPECTS" (ISBN: 0 444 87706 1). Książka ta jest zbiorem materiałów identycznie zatytułowanej konferencji roboczej grupy IFIP WG 8.1, a jej publikacja, zgodnie z praktykowanym od wielu lat zwyczajem, została ściśle zsynchronizowana z terminem obrad tej konferencji (Barcelona, 16—18 kwietnia 1985 r.).

Współczesne systemy informacyjne w sposób bardzo wyraźny rozwijają się w kierunku architektur rozproszonych, automatyzacji biur i elektronicznej poczty oraz systemów wspomagania decyzji, popularnie zwanych systemami ekspertowymi. Ewolucja ta wymaga ciągłego rozwoju dotychczasowych metod, a także poszukiwania nowych podejść do problematyki szeroko rozumianego modelowania danych oraz działania systemów informacyjnych.

Książka liczy 236 stron i zawiera teksty 15 referatów. 28-osobowy komitet programowy oraz trzech redaktorów (J. Bubenko, A. Sernadas i A. Olive) dokonali wyboru spośród 71 zgłoszonych referatów. Tak ostra selekcja materiałów wpłynęła niewątpliwie w sposób decydujący na wysoki poziom naukowy omawianej publikacji.

Treść książki podzielono na pięć części. W części pierwszej, zatytułowanej „Systems and Languages", Mark i Roussopoulos w pracy „The New Database Architecture Framework" przedstawiają nową koncepcję architektury bazy danych, opartą na stosunkowo dobrze znanych pojęciach samoopisywania i samodokumentowania baz danych. Demo, Deliva i Giolito w pracy „An Entity-Relationship Query Language" przedstawiają rozwiązanie w modnym obecnie kierunku, jakim są języki zapytań do baz danych wykorzystujących model obiektowo-związkowy Chen'a (E--R). Część tę kończy praca Andersona i Claghorna „ADE: mapping between the external and conceptual levels". Przedstawiono w niej środowisko rozwoju zastosowań (Application Development Environment), charakteryzujące się udostępnianiem w trybie podziału (ang. sharing) nie tylko wartości danych, lecz także ich rozmieszczenia, sposobów formatowania i konwersji reprezentacji.

Części 2, 3 i 5 książki noszą tytuły „Modeling Approaches". Niezwykle ważne zagadnienie modelowania czasu w systemach informacyjnych porusza praca Richtera „Clocks and Their Use for Time Modeling". Istotne novum proponowanego podejścia polega na tym, że do systemu opisanego siecią Petri wprowadza się czas przez wkomponowanie w strukturę sieci generatorów impulsów (zegary). Z kolei Schiel w pracy „The time dimension in information systems" wstępnie rozróżnia abstrakcyjne obiekty przestrzenne i czasowe oraz proponuje jednolite podejście do czasowej charakterystyki obiektów realnych, związków i wydarzeń. Morgenstern w pracy „Connections Among Application Based Objects in Directed Edge-Labelled Hypergraphs" wprowadza grafy skierowane o zaetykietowanych krawędziach jako metodę ujmowania informacji o jawnie określonych nazwach powiązań i zależności danych. Zaproponowane grafy są de facto nazwanymi odwzorowaniami pomiędzy zbiorami atrybutów. Elementy teorii grafów wykorzystuje także Pletat w pracy „A graphtheoretic semantic for semantic data models". Tym razem chodzi o definiowanie semantyki języka definicyjno-mani-

pulacyjnego, wykorzystującego wspomniane już wcześniej podejście obiektowo-związkowe. Horndasch, Studer i Yasdi w pracy „An Approach to Conceptual Schema Design of Information Systems" proponują nową metodologię projektowania tzw. schematów pojęciowych, która nie zaniedbując klasycznych aspektów modelowania danych podkreśla modelowanie dynamiki projektowanego systemu informacyjnego. Najważniejszym narzędziem procesu projektowania jest sieć wysokiego poziomu TJM (Temporal Hierarchic Data Model). W pracy Janssona „A Framework for Representation" rozważane są ogólne zagadnienia schematów pojęciowych, nawiązujące w pewnym stopniu do podanych przez Smith'ów klasycznych już idei agregacji i generalizacji.

Bracchi i Pernici w pracy „Specification of Control Aspects in Office Information Systems" definiują język formalny do specyfikacji reguł sterowania, zaprojektowany zgodnie z wymogami opracowanego wcześniej przez ten sam zespół modelu SOS (Semantic Office System). Autorzy rozważając zintegrowane sterowanie deterministyczne w systemie informacyjnym o orientacji biurowej analizują szczegółowo funkcje i potrzebne do tego celu operatory czasowe. Flint i Leveson w pracy „The PAMIS Approach to Modeling Database Activity" zajmują się zagadnieniami modelowania i oceny aktywności systemu bazy danych (PAMS — Prototype Activity Modeling System).

Część czwarta książki, zatytułowana „Logical Frameworks", otwiera praca Lipecka, Ehricha i Gogolla „Specyfying Admissibility of Dynamic Database Behaviour using Temporal Logic". W pracy tej autorzy proponują stosowanie logiki temporalnej jako formy rachunku do wyrażania tzw. więzów integralności (ang. integrity constraints), określających dostępność rozpatrywanego dynamicznie systemu bazy danych. Carmo w pracy „The INFOLOG Branching Logic of Events" przedstawia nieliniową logikę temporalną jako środek abstrakcyjnej specyfikacji słabo związanych (ang. loosely connected) systemów informacyjnych. Praca Veloso i Furtado „Towards simpler and yet complete formal specifications" dotyczy nowej metody formalnej specyfikacji systemu informacyjnego z bazą danych, charakteryzującą się wybitną prostotą opisu. Autorzy przedstawiają również prototyp napisanego w PROLOGU odpowiedniego oprogramowania narzędziowego. Ostatnia w tej części praca Kluga „A Tableaux Approach for Consistency Checking" zawiera ogólną koncepcję specyfikacji i weryfikacji modeli pojęciowych w systemach informacyjnych. W pracy tej podano metodę sprawdzania spójności, której istota sprowadza się do analizy, czy zostały spełnione zadane statyczne i czasowe ograniczenia systemu.

Należy stwierdzić, że wyżej scharakteryzowana publikacja jest bez wątpienia pozycją bardzo wartościową i tym samym godną polecenia. Jej treść może być szczególnie interesująca i użyteczna dla tych informatyków, którzy pracują nad zagadnieniami metodologii projektowania systemów informacyjnych oraz baz danych.

PIOTR J. JASIŃSKI
MICHAŁ SAJKOWSKI

dokończenie ze str. 14

wstępnego z docelową kompilacją programu. Pewien kłopot powodują ograniczenia w zasobie znaków na ODRZE 1305. Mimo zawężenia kodu do 64 znaków ICL 1900 można jednak poradzić sobie z transkrypcją nazw operatorów.

Minikomputer SM-4 jako środowisko sprzętowe stanowi bardziej naturalny grunt dla implementacji RATFORU niż ODRA 1305. Sprzyja temu zgodność kodu znaków i rozwiązania z zakresu zarzadzania plikami, zastosowane w systemach operacyjnych tego komputera. Warto dodać, że RATFOR ma swoje mutacje na minikomputerach firmy DEC.

Analiza współczesnych ofert w zakresie oprogramowania mikrokomputerów wskazuje, że typowy zestaw translatorów składa się z BASICA, FORTRANU (rzadko FORTRANU 77), PASCALA, COBOLU i — niekiedy — języka C. Rozpowszechnienie RATFORU zależy — jak sądzę — od wzajemnych proporcji w ilości oprogramowania w językach FORTRAN i PASCAL.

* * *

Możliwość pisania o RATFORZE, a także jego zainstalowania, autor zawdzięcza życzliwości i osobistemu zaangażowaniu profesora Ralpha E. Griswolda z Wydziału Informatyki Uniwersytetu w Tucson, stan Arizona (USA).

LITERATURA

[1] Computing Reviews, Vol. 17, No. 6, June 1976, notka nr 29939
[2] Griswold R. E.: Portable Icon Distribution Package, Version 2.0, Department of Computer Science, University of Arizona, March 1983
[3] Griswold R. E., Griswold M. T.: The Icon Programming Language. Prentice-Hall, Englewood Cliffs, NJ, 1983
[4] Kernighan B. W.: Software Tools in PASCAL. Addison-Wesley Publ. Co., Reading, MA, 1982
[5] Kernighan B. W., Plauger P. J.: Software Tools. Addison-Wesley Publ. Co., Reading, MA, 1976
[6] Lecarme O. L.: Computing Reviews, Vol. 24, No. 5, May 1983, notka nr 40285
[7] Russel D. L.: Computing Reviews, Vol. 18, No. 11, November 1977, notka nr 32153
[8] Samek W. J.: Computing Reviews, Vol. 18, No. 5, May 1977, notka nr 31358.

Konferencje AFCET

Stowarzyszenie francuskich informatyków AFCET (Association Francaise pour la Cybernetique, Economique et Technique) organizuje w 1986 r. następujące dwie międzynarodowe konferencje:

EKONOMIA A SZTUCZNA INTELIGENCJA („Economics and Artificial Intelligence") w terminie od 2 do 4 września 1986 r. w Aix en Provence, Francja.

Tematyka konferencji:

A) metody i rozwiązania eksperymentalne
- nowe metody modelowania złożonych systemów
- inteligentny system informacyjny badań ekonomicznych
- rozwój specjalistycznego sprzętu komputerowego i oprogramowania

B) zastosowania
- systemy wspomagania decyzji (DSS) na potrzeby zarządzania i organizacji
- inteligentne modele prognozowania
- inteligentne kierowanie działalnością administracyjną
- modelowanie wielkich systemów
- systemy diagnostyczne.

VIII MIĘDZYNARODOWA KONFERENCJA TECHNIKI ROZPOZNAWANIA OBRAZÓW („Eight International Conference on Pattern Recognition") w terminie od 28 do 31 października 1986 r. w Paryżu.

Sponsorami konferencji są: międzynarodowe stowarzyszenie techniki rozpoznawania obrazów (IAPR) oraz rząd i przemysł francuski.

Tematyka konferencji:

- klasyfikacja danych i algorytmy grupowania
- przetwarzanie sygnałów i obrazów
- analiza odwzorowań i rozpoznawanie cech
- zakłócenia odwzorowań, indukcyjne metody percepcji
- modelowanie percepcji człowieka
- interpretacja i rozpoznawanie obrazów i mowy
- komputerowizja i systemy ekspertowe w dziedzinie rozpoznawania obrazów
- specjalizowane układy VLSI do rozpoznawania obrazów
- zastosowania przemysłowe
- robotyka
- biomedyczne rozpoznawanie obrazów
- zdalne sondowanie
- rozpoznawanie obrazów w automatyzacji prac biurowych
- interpretacja i weryfikacja tekstów.

Bliższych informacji udziela AFCET, 156, Boulevard Péreire F. 75017 Paris, tel. 766-24-19, telex 290 163 EURTEL Code 235.

Szkaradnik Z.: Porównanie języków programowania mikrokomputerów

INFORMATYKA 1985, nr 11—12, s. 2

Prezentacja wyników porównawczego testowania translatorów języków FORTRAN, PASCAL, C, BASIC oraz FORTH, przeprowadzonego przez autora artykułu na mikrokumputerze MERA 60 celem zbadania efektywności tych języków.

Bielecki J.: Programowanie asemblerowe w języku fig-FORTH

INFORMATYKA 1985, nr 11—12, s. 6

Szczegółowe omówienie sposobu definiowania nowych operatorów i instrukcji języka FORTH za pomocą sekwencji rozkazów maszynowych.

Gasik S.: Wprowadzenie do programometrii

INFORMATYKA 1985, nr 10, s. 9

Charakterystyka metod określania i pomiaru złożoności oprogramowania. Omówiono czynniki wpływające na złożoność oprogramowania, a także zdefiniowano pojęcie programometrii oraz podano jej cele i zastosowania.

Płoski Z.: RATFOR — czyli strukturalny FORTRAN (2)

INFORMATYKA 1985, nr 11—12, s. 13

Druga część charakterystyki języka RATFOR, zawierająca omówienie pozostałych instrukcji, opinię o języku oraz podstawowe zasady jego implementacji.

Bieleninik E., Hudyma E., Kosmulska-Bochenek E., Lewoc J., Stanisz A.: Podsieć komunikacyjna Międzyuczelnianej Sieci Komputerowej

INFORMATYKA 1985, nr 11—12, s. 15

Szczegółowa charakterystyka założeń oraz przyjętych rozwiązań podsieci komunikacyjnej Międzyuczelnianej Sieci Komputerowej. Omówiono strukturę, zakres usług i obsługę operatorską podsieci oraz uzyskane doświadczenia i plany dalszego rozwoju.

Szkaradnik Z.: A comparison of microcomputer programming languages

INFORMATYKA 1985, No. 11—12, p. 2

Results of FORTRAN, PASCAL, C, BASIC and FORTH translators testing, performed by the author on MERA 60 microcomputer to investigate their effectivness.

Bielecki J.: Assembly programming in the fig-FORTH

INFORMATYKA 1985, No. 11—12, p. 6

Detailed discussion of the definition method for new operators and statements using machine instruction sequences.

Gasik S.: Introducing to programometry

INFORMATYKA 1985, No. 10, p. 9

Characteristics of software complexity defining and measuring methods. Factors influencing on software complexity are discussed, as well as the idea of programometry is defining and its aims and applications are presented.

Płoski Z.: RATFOR — or structured FORTRAN (2)

INFORMATYKA 1985, No. 11—12, p. 13

Second part of characteristics of the RATFOR programraing language, including the discussions of further statements, the assessment of the language and basic rules for the implementation.

Bieleninik E., Hudyma E., Kosmulska-Bochenek E., Lewoc J., Stanisz A.: Communication subnetwork of the MSK-Interuniversity Computer Network

INFORMATYKA 1985, No. 11—12, p. 15

Detailed characteristics of assumptions and solutions of the communication subnetwork in the MSK-Computer Network. Structure, services and operation, as well as experience and development plans are discussed.

Шкарадник З.: Сравнение языков программирования мини-ЭВМ

INFORMATYKA 1985, № 11—12, стр. 2

Обсуждено результаты сравнительных испытаний трансляторов языков FORTRAN, PASCAL, C, BASIC, FORTH проведенных авторов на мини-ЭВМ MERA 60 с целью исследования их эффективности.

Белецки Я.: Программирование на языке fig FORTH

INFORMATYKA 1985, № 11—12, стр. 6

Подробно обсуждено способы определения новых операторов и инструкции языка FORTH с помощью последовательностей машинных команд.

Гасик С.: Введение в программометрию

INFORMATYKA 1985, № 10, стр. 9

Характеристика методов определения и измерен я сложности программного обеспечения. Обсуждено факторы влияющие на сложность программ, дано определение понятия программометрии и представлено ее цель и применение на практике.

Плоски З. RATFOR — структурный FORTRAN (2)

INFORMATYKA 1985, № 11—12, стр. 13

Вторая часть характеристики языка RATFOR содержащая обсуждение инструкций, мнение о языке и основные правила его реализации.

Беленинк Э., Худыма Э., Космульска-Бохенек Э., Левоц Ю., Станиш А.: Система связи междучилищной вычислительной сети

INFORMATYKA 1985, № 11—12, стр. 15

Подробная характеристика предположений и принятых решений в системе связи междучилищной вычислительной сети. Обсуждено структуру, возможные услуги и работу персонала обслуживающего сеть, а также накопленный опыт и планы расширения сети.

Szkaradnik Z.: Ein Vergleich der Programmiersprachen von Mikrorechnern

INFORMATYKA 1985, Nr. 11—12, S. 2

Ergebnisse eines Testes von FORTRAN, PASCAL, C, BASIC und FORTH Übersetzern, der zwecks Leistungsprüfung dieser Sprachen vom Autor auf MERA 60 Mikrorechner durchgeführt wurde.

Bielecki J.: Assemblerprogrammierung in der fig-FORTH-Sprache

INFORMATYKA 1985, Nr. 11—12, S. 6

Definierungsmethode der neuen Operatoren und Anweisungen in der FORTH-Programmiersprache mit Hilfe der Maschinensprachesequenzen.

Gasik S.: Einführung zur Programmometrie

INFORMATYKA 1985, Nr. 10, S. 9

Eine Charakteristik von Methoden für Bezeichnung und Messung der Softwarekompliziertheit. Es werden Faktoren, die auf Softwarekompliziertheit beeinflussen, besprochen, sowie das Wesen der Programmometrie definiert und ihre Ziele und Anwendungen angegeben.

Płoski Z.: RATFOR — das heisst struktureller FORTRAN (2)

INFORMATYKA 1985, Nr. 11—12, S. 13

Zweiter Teil einer Charakteristik von RATFOR-Sprache, die eine Besprechung von übrigen Anweisungen, eine Beurteilung der Sprache und Grundsätze ihrer Implementierung umfasst.

Bieleninik E., Hudyma E., Kosmulska-Bochenek E., Lewoc J., Stanisz A.: Kommunikationsunternetz des MSK-Hochschulrechnernetzes

INFORMATYKA 1985, Nr. 11—12, S. 15

Eine detalierte Charakteristik von Voraussetzungen und Lösungen des Kommunikationsnetzes im MSK-Hochschulrechnernetz. Es wurden Struktur, Dienstleistungen und Bedienung des Unternetzes, sowie Erfahrungen und Entwicklungspläne besprochen.

ZE ŚWIATA

	nr	str.
Świat superkomputerów CRAY — **Beniamin Gołdys, Wacław Iszkowski**	1	26
Najnowsze tendencje w dziedzinie komputerowych systemów sterowania — **Janusz Zalewski**	2	22
Rodzaje drukarek — oprac. **Teresa Wilczek**	2	28
NEW GENERATION COMPUTING — **W.E.B.**	2	29
Naukowiec — biznesmenem — oprac. **W. K.**	3	23
Drukarka strumieniowa HP — oprac. **J. T.**	3	24
Sztuczna inteligencja i systemy eksportowe — oprac. **Hanna Kontkiewicz**	3	26
Pozorne złagodzenia — **Adam B. Empacher**	3	27
Jak zapewnić ochronę danych? — tłum. **Zofia Szewc**	4	25
Informatyka a suwerenność — oprac. **A.B.E.**	4	26
Mikrokomputery w automatyce i technice systemów — **Wacław Iszkowski**	4	27
O robotyce — **A.B.E.**	4	27
SICOB'84 — oprac. **Hanna Kontkiewicz-Chachulska**	5	26
PICK — tajemnica informatyki — oprac. **Wacław Iszkowski**	5	28
Dyski laserowe — oprac. **M. G.**	6	25
Dyski 3M — oprac. **Teresa Wilczek**	6	25
Zastosowanie informatyki w WRL — oprac. **M. Siedlecki**	7	24
Aktualne problemy ochrony oprogramowania — oprac. **Ryszard K. Kott**	7	25
Nowości w brytyjskiej mikroinformatyce — oprac. **Jan Ryżko**	7	27
Komputer roku 2000 — oprac. **Ryszard Kurzyjamski**	7	28
Czechosłowacki program elektronizacji — **W.K.**	7	28
Japońska elastyczność — oprac. **K.I.**	7	29
Produkcja oprogramowania w krajach RWPG — oprac. **Swietłana Asmołowa**	8	25
Zwiększenie pojemności kostek pamięciowych — oprac. **Jan Ryżko**	8	27
Nagroda czasopisma ELEKTRONICS WEEK — oprac. **Jan Ryżko**	8	29
Zamykanie głów na klucz — **A.B.E.**	8	29
Informatyka na Wiosennych Targach Lipskich — **Władysław Klepacz**	9	30
Producenci komputerów 8-bitowych w opałach — **Ryszard K. Kott**	10	24
Firmy brytyjskie wchodzą na rynek krajów socjalistycznych — **Ryszard K. Kott**	10	25
Targi Hanowerskie — **Władysław Klepacz**	10	26
Najnowsze tendencje w dziedzinie komputerowych systemów sterowania — problemy oprogramowania — **Janusz Zalewski**	11—12	45

RECENZJE

Zastosowanie mikroprocesorów w energoelektronice — **Janusz Zalewski**	1	30
Programowanie operacji wejścia-wyjścia — **Wacław Iszkowski**	3	28
Bezpośrednie systemy informacyjne — **Andrzej Sokołowski**	10	27
Dla kogo ten „podręcznik"? — **Jerzy Kucharczyk, Andrzej Niemiec**	10	28
Systemy informacyjne: aspekty teoretyczne i formalne — **Piotr J. Jasiński, Michał Sajkowski**	11—12	50

TERMINOLOGIA

Słownik terminów związanych z arytmetyką zmiennoprzecinkową — **Zbigniew Banasik**	1	30
Słownik pojęć i terminów z dziedziny grafiki komputerowej — **Wojciech Mokrzycki, Lena Walkiewicz**		
Część 1	2	30
Część 2	3	29
Część 3	4	29
Uzupełnienie — **Wojciech Mokrzycki**	8	30
Nasz drugi język — **Maciej M. Sysło**	4	30
Uwagi o terminologii języka ADA — **Jan Bielecki**	5	29
Problemy tworzenia terminologii — **Władysław Klepacz**	6	26
Rozwój terminologii języka ADA — **Janusz Zalewski**	7	30
Szanowna Redakcjo — **J. Jankowski**	8	31
Nazwy własne produktów informatycznych — **Wacław Iszkowski**	9	27
Zmienno- i stałoprzecinkowy czy zmienno- i stałopozycyjny? — **Janusz Zalewski**	10	29
DIXIT ALGORIZMI czyli o pochodzeniu i znaczeniach słowa algorytm — **Janusz Stokłosa**	11—12	48

LISTY

Liczyć rozumniej — **Maciej M. Sysło**	4	28
Skoordynować — **Halina Towarnicka**	4	28
Mówić o sobie — **Barbara Fabiańska**	5	30

POGLĄDY

Mikrokomputery i nauczyciele — **Emanuel Czyżo**	4	III okł.
Nożyce norymberskie — **Jakub Tatarkiewicz**	5	III okł.

KONFERENCJE

Sieci komputerowe — usługi, protokoły, modele — 22—23 październik 1985	3	27
Mikrokomputer '85 — 24—27 wrzesień 1985	3	28
INFOPROD '85 — Techniki komputerowe w zarządzaniu produkcją — 3—5 październik 1985	5	31
FTSD '85 — Systemy tolerujące uszkodzenia i diagnostyka — 10—12 wrzesień 1985	6	24
Problemy komputeryzacji projektowania urządzeń elektroenergetycznych — 17—19 wrzesień 1986 r.	6	28
II Szkoła — Mikrokomputery 16-bitowe — 9—11 grudzień 1985 r.	7	4
CAMAC-86 — 22—24 kwietnia 1986 r.	8	31
Zastosowanie mikroprocesorów w automatyce i pomiarach — 25 wrzesień 1986 r.	8	31
Systemy Zarządzania Bazą Danych (NRD) — 30 listopada — 5 grudnia 1986 r.	8	31

GRAFIKA

Grafika w INFORMATYCE — **Jakub Tatarkiewicz**	1	III okł.
Grafika — **Rafał Pietrak, Jakub Tatarkiewicz**	1	IV okł.
Fraktale — **Rafał Pietrak, Jakub Tatarkiewicz**	2	IV okł.
Spirale i labirynt — **Zbigniew Szkaradnik**	3	IV okł.
Kwiaty — **Ewa Grabska, Wojciech Grabski**	4	IV okł.
Grafika — **Piotr Zielczyński**	5	IV okł.
Fraktale — **Rafał Pietrak, Jakub Tatarkiewicz**	7	IV okł.
Grafika — **Zenon Jędrzykiewicz**	10	IV okł.

■ Sprawna organizacja wielkich zawodów, takich jak np. mistrzostwa lekkoatletyczne, jest już obecnie nie do pomyślenia bez pomocy komputerów. Wykorzystywano je do tej pory głównie jednak do obsługi olbrzymiej bazy danych, dotyczącej zawodników, aktualnych wyników itp.

Podczas tegorocznego maratonu w Londynie organizatorzy poszli dalej — postanowiono bowiem wypróbować automatyczny system rejestracji czasu na mecie. Maraton ten jest gigantyczną imprezą rozgrywaną zwykle w drugiej połowie kwietnia. W tym roku wzięło w nim udział ok. 22 000 osób, wybranych na podstawie ok. 80 000 zgłoszeń. Oczywiście komputer ułatwia również stronę czysto organizacyjną tej imprezy.

Rejestrując wyniki na mecie, powszechnie stosuje się system półautomatyczny, wspomagany komputerem. Polega on na tym, że zamiast tradycyjnego stopera, sędziowie trzymują w ręku „elektroniczny stoper", będący wyspecjalizowaną końcówką. Końcówka, połączona kablem z komputerem obsługującym, składa się z klawiatury służącej do wprowadzenia numeru zawodnika oraz z przycisku czasowego, który sędzia naciska w chwili, gdy zawodnik mija metę — tak samo jak w zwykłym stoperze. Czas jest automatycznie rejestrowany przez komputer, a stopera można natychmiast użyć do pomiaru czasu kolejnych sportowców. Oczywiście organizatorzy korzystają również z zapisu video, gdyż w szczytowym punkcie zakończenia biegu na metę wypadają jednocześnie całe tłumy zawodników.

Ten „przestarzały" system postanowiono usprawnić przez zastosowanie czytników kodu kreskowego (ang. bar code), takiego jak stosuje się powszechnie na Zachodzie do oznaczania towarów. Próby nad wykorzystaniem tego systemu przeprowadzono podczas ubiegłorocznego konkursu maratonu, kiedy to niektórzy zawodnicy otrzymali plakietki z kodem kreskowym. Zainstalowany automatyczny czytnik kodu wykrywał obecność plakietki i dokonywał odczytu zapisanego na niej numeru w chwili, gdy zawodnik mijał metę. Tegoroczny przełom polegał na tym, że po raz pierwszy na świecie postanowiono objąć tym systemem wszystkich uczestników biegu. Oczywiście utrzymano również półautomatyczny system rejestracji wyników.

Z punktu widzenia organizatorów maratonu główne zalety stosowania podwójnego systemy rejestracji to: praktycznie całkowite wyeliminowanie pomyłek sędziowskich, a także skrócenie czasu opracowywania kompletu wyników (z jednego tygodnia do dwóch dni — oficjalne wyniki pierwszego tysiąca zawodników po raz pierwszy mogą ukazać się jeszcze w dniu zawodów!).

System komputerowy obsługujący tegoroczny maraton zainstalowany był w przyczepie kempingowej, zaparkowanej na moście westminsterskim, tj. w pobliżu mety. Składał on się z dwóch minikomputerów VAX11/730, pracujących w sieci, oraz z połączonych z nimi dwóch mikrokomputerów MicroVAX1, które obsługiwały sieć „stoperów elektronicznych" firmy SEIKO. Czytniki kodu kreskowego połączone były bezpośrednio z maszynami 11/730, które obsługiwały też zwykłe terminale używane przez organizatorów i sędziów, a także główną bazę danych zawodów, na standardowych dyskach sztywnych. (RKK)

*

■ HOLOLINK — holograficzny system połączeń wieloprocesorowych umożliwia przesyłanie danych z prędkością 10^{14} bitów/s, co wielokrotnie przewyższa prędkości osiągane dotąd w tradycyjnych połączeniach elektrycznych. System opracowany został przez Fraunhofer Gesellschaft, wielką zachodnioniemiecką organizację prowadzącą badania stosowane. Koncepcja połączeń holograficznych polega na wykorzystaniu hologramu przestrzennego, zamiast tradycyjnego systemu łącz elektrycznych. Przesyłanie sygnałów polega na emitowaniu promieni światła w jednym układzie fotodiod i odpowiednim oświetlaniu tymi promieniami drugiego układu. Jeden zespół fotodiod połączony jest bezpośrednio z systemem wieloprocesorowym, natomiast drugi współpracuje z bazami danych. Na drodze promieni świetlnych umieszczony jest hologram przestrzenny zwany pamięcią holograficzną. Hologram ten pełni rolę układu sterującego. Zespół fotodiod układu wieloprocesorowego podzielono na jednakowe segmenty, a każdy z segmentów połączony jest z jednym procesorem. Liczba fotodiod w segmencie odpowiada liczbie baz danych. Tak więc każdy emitowany, w układzie wieloprocesorowym, promień świetlny informuje jednoznacznie o zapotrzebowaniu określonej bazy danych przez dany procesor.

Pamięć holograficzna, załamując odpowiednio promienie świetlne, skierowuje je na fotodiody baz danych. Połączenia holograficzne pozwalają na jednoczesny dostęp kilku procesorów do tych samych danych. Prawidłową organizację pracy może w tym wypadku zapewnić dodatkowy procesor komunikacyjny, zarządzający kolejkami zgłoszeń. Praktyczne rozwiązanie tego problemu znajduje się obecnie w stadium realizacji.

Specjaliści uważają, że system HOLOLINK może znacznie zwiększyć efektywność działania wszelkich systemów komputerowych, o architekturze różnej od klasycznej architektury von Neumanna. (TW)

*

■ W ośrodku obliczeniowym uniwersytetu w Hamburgu (RFN) otrzymano na dużym komputerze SIEMENS 7.882 nową liczbę pierwszą $5 \times 2^{23473} + 1$, której pełny zapis obejmuje 7067 cyfr. To rekordowe osiągnięcie ma również znaczenie praktyczne, ponieważ często ze względu na trudność deszyfracji liczby pierwsze używane są do kodowania poufnych informacji. (WK)

W skrócie • W skrócie • W skrócie • W skrócie • W skrócie • W skrócie

WARUNKI PRENUMERATY NA 1986 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu" rozszerzonym dla potrzeb Wydawnictwa o część dotyczącą zamówienia. Blankiety te będą dostarczane przez Zakład Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.

Wpłacać należy na konto NBP III O/M Warszawa 1036-7430-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:

— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

Uwaga!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 oraz w Dziale Handlowym Wydawnictwa ul. Bartycka 20 skr. poczt. 1004, 00-950 Warszawa, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 120 zł za numer (35 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
360	105	720	210	1440	420

■ Programować można też dotykiem. Firma CARROL TOUCH produkuje różnorodne systemy monitorowe programowane metodą dotykową. Systemy te nie wymagają nauki rozkazów języka ani wciskania przycisków — wystarczy jedynie dotykać odpowiednich miejsc ekranu monitora. Przystosowany do potrzeb użytkownika, monitorowy system graficzny w trybie pytań i wybieranych dotykiem odpowiedzi realizuje odpowiednie funkcje. Urządzenie może być pomocne w kreśleniu projektów technicznych, w sterowaniu procesami przemysłowymi, w dydaktyce, wojskowości, w bankach informacji, a także w pracach biurowych. Firma dostarcza sprzęt i oprogramowanie monitora dotyku przystosowane do potrzeb użytkownika. (TW)

•

■ Rośnie liczba urządzeń umożliwiających komunikację z komputerem za pomocą głosu. Do tej pory nie była to jednak sprawa łatwa — należało mówić powoli, wyraźnie akcentując sylaby. W rezultacie przystawki głosowe, mające ułatwiać m.in. wprowadzanie danych były niewygodne i męczące w dłuższym użytkowaniu. Najnowsze urządzenie, przystawka firmy VERBEX, nie tylko likwiduje te niedogodności, lecz również „rozumie" mutacje wymowy czy akcentowania i nie wymaga pomieszczeń dźwiękoszczelnych. Urządzenie pomyślane jest jako system wspomagający gospodarkę materiałową, a zwłaszcza inwentaryzowanie i kontrolę zapasów oraz różnego rodzaju testy. VOICE RECOGNIZER SERII 40000 może współpracować zarówno z dużymi maszynami, jak i z mini- i mikrokomputerami. Urządzenie wyposażone jest w zbiór wzorców głosowych i słownik zastosowań, wpisany do pamięci sprzężonego komputera lub do własnej przystawki pamięci CMOS. (TW)

•

■ Pod kierownictwem Mike'a Kamińskiego z koncernu GENRAL MOTORS (GM) opracowywano w ostatnim pięcioleciu system MAP (Manufacturing Automation Protocol) dla zakładu produkującego koła zębate do skrzyni biegów w Saginaw (stan Michigan). Zakłady te, reklamowane jako „fabryka jutra" będą miały — jako pierwsze w koncernie GM — całkowicie skomputeryzowane automatyczne linie produkcyjne. Z technicznego punktu widzenia system MAP — stanowi rozbudowaną sieć lokalną typu kablowego przystosowaną, dzięki nawiązaniu do standardów ISO, do współpracy z różnorodnym sprzętem informatycznym, nawet różnych producentów. Uruchomienie systemu MAP w Saginaw przewidywane jest we wrześniu br. Uprzednio prototyp przemysłowy (BETA według terminologii amerykańskiej) został przebadany eksperymentalnie w detroickiej montowni w zakresie prac lakierniczych wykonywanych przez roboty. W roku przyszłym koncern GM przewiduje zainstalowanie systemu MAP w pięciu dalszych zakładach, a w 1987 r. w kilkudziesięciu następnych. Szczegóły systemu zostaną ujawnione publicznie na tegorocznej jesiennej wystawie przemysłowej AUTOFACT.

System MAP w Saginaw składa się z 50 tzw. ogniw, z których każde obejmuje 6—7 różnych urządzeń sterujących obrabiarkami lub komorami technologicznymi. Całość jest obsługiwana przez lokalny komputer realizujący funkcje koordynacyjne i współpracujący z dużym komputerem. Istotną cechą MAP jest to, iż stanowi on mikroprocesorową realizację funkcji protokolarnych przewidzianych standardem otwartego systemu połączeń — Open System Interconnection (OSI) — zalecanego przez ISO (International Standards Organization). Co więcej, system MAP wprowadza dalsze standardy, które mają szansę stać się zaleceniami międzynarodowymi. Wersja MAP-2.1 jest pierwszym z serii proponowanych standardów, jakie w swych planach automatyzacji produkcji przyjęły m.in. firmy VOLVO, FIAT, PHILIPS i RENAULT. Jednakże 17 innych firm europejskich postanowiło podjąć w ramach projektu ESPRIT własny system standardów, nazwany CIM (Computer Integrated Manufacturing). Opracowanie mikroprocesorowej architektury tego systemu będzie wymagać około 1000 osobolat pracy, ale można oczekiwać, że w zakresie OSI zrealizuje ona szerzej zalecenia ISO aniżeli wspomniany MAP. Systemem CIM interesują się zwłaszcza koncerny GEC oraz BRITISH AEROSPACE, natomiast nie przejawia tu zainteresowania firma ICL. (ABE)

•

■ Aktualny rekord świata w programowaniu gry szachowej należy do pracowników firmy CRAY RESEARCH. Najnowszy superkomputer tej firmy, czteroprocesorowy XMP/48, jest w stanie w ciągu 90 sekund rozważyć około 7 mln pozycji debiutowych odpowiadających 7—8 ruchom naprzód. W grze końcowej jest nawet w stanie analizować sytuacje 45-ruchowe. (ABE)

•

Wiosną br. koncern IBM zapowiedział wprowadzenie — na razie na rynek amerykański — nowej rodziny komputerów SYSTEM/88. Składają na nią się trzy procesory oznaczane jako 4576-20, 4576-40 i 4576-60, o orientacyjnej cenie od 150 tys. dolarów wzwyż. Procesory te stanowią licencyjną kopię rodziny STRATUS-32, która odniosła sukces rynkowy, jako szczególnie odporna na awarie — dzięki wbudowanym układom zastępczym na najniższym poziomie realizacji technicznej. (ABE)

•

■ Ostatnio pewien rozgłos w Wielkiej Brytanii uzyskał żłobek Politechniki Lancasterskiej w Preston. Jest to pierwszy żłobek, w którym maluchy bawią się mikrokomputerami (ZX SPECTRUM+). Pomimo podobnych, miłych doświadczeń z przedszkolakami z klas zerowych, kierowniczka, p. Christine Cross, była zaskoczona jak łatwo i szybko dzieci nauczyły się rozpoznawać klawisze używane w rozmaitych grach edukacyjnych. Mikrokomputer działa bez awarii już dłuższy czas, mimo że średnia wieku użytkowników wynosi... dwa i pół roku. Oczywiście maluchy przepadają za tą zabawką — niektóre chciałyby się bawić cały dzień! Jednak ze względów pedagogicznych dzienna dawka nie przekracza 10—15 minut na jednego siusiumajtka. Ech, chciałaby dusza do raju! (RKK)

•

■ Sądząc po tym, co wydarzyło się niedawno w Omaha, stolicy amerykańskiego stanu Nebraska, lepiej nie myśleć co będzie się działo w XXI wieku.

Otóż zaczęło się od tego, że technik pełniący nocny dyżur w centrali telefonicznej zauważył niezwykle obciążenie niektórych linii i zlokalizował jego źródło w wielkim biurowcu, nieczynnym oczywiście o tej porze. Zaalarmowana policja przeprowadziła sprawną, natychmiastową akcję, która zakończyła się pełnym sukcesem — złoczyńców zdybano na gorącym uczynku telefonowania. Winowajcami okazały się... dwa automaty do coca-coli. Maszyny te wyposażono w mikrokomputerowy układ sterowania, który w przypadku wyczerpania się coli w pojemniku, dzwoni pod specjalny numer, zgłaszając konieczność dostawy. W przypadku, gdy numer nie odpowiada, mikrokomputer dzwoni ponownie. Okazało się, że obie maszyny usiłowały zgłosić zapotrzebowanie przez co najmniej 12 godzin. (RKK)

•

■ Po kostkach pamięci RAM o pojemności 64 K bitów do masowej produkcji wchodzą już kostki 256 K bitów. Jest to oczywiście pamięć dynamiczna (DRAM). W tej chwili cena za jedną kostkę kształtuje się na poziomie niewiele ponad 10 dolarów USA, tzn. spadła w ciągu ostatnich dwu lat o połowę. Prawdopodobnie pierwszym producentem, który uruchomi masową produkcję, jest firma TEXAS INSTRUMENTS (oddział TEXAS JAPAN). Zakłady w Miho k.Tokio są na etapie rozruchu, z docelową produkcją rzędu 4 mln sztuk miesięcznie. Wielcy producenci japońscy nieznacznie pozostają w tyle — NEC, FUJITSU, HITACHI i MITSUBISHI zapowiedziały uruchomienie produkcji masowej w tym roku. MITSUBISHI podała również, że zamierza w niedługim czasie uruchomić produkcję kostek statycznych o pojemności 256 Kbitów oraz próbną produkcję kostek 1 Mbit DRAM. Pierwsze kostki 1 Mbit DRAM wyprodukowała już także firma AT & T (USA).

Prace rozwojowe nad nowymi technologiami produkcji stają się coraz bardziej kosztowne, wymagają ponadto coraz nowocześniejszego oprzyrządowania. Nic dziwnego, że powoli zaczyna to przekraczać możliwości finansowe poszczególnych koncernów. Tak np. japoński NIPPON TELEGRAPH ujawnił plany badawcze, zmierzające do uzyskania kostek 64 Mbitów DRAM — jednak dla urzeczywistnienia takiego programu trzeba m.in. zakupić... synchrotron (koszt ok. 25 mln dol.). Wszystko to dotyczy cyzelowania technik dość dobrze już znanych, nie widać natomiast specjalnego zapału do inwestowania w badania nad technologiami waflowymi (WSI — Wafer SI). (RKK)

CSK—Computer Studio Kajkowscy

81-505 GDYNIA ORŁOWO, ul. Balladyny 3B, tel. 29-00-18

Komputer osobisty może być przydatny niemal na każdym stanowisku pracy. Wymaga jednak odpowiedniego oprogramowania użytkowego. W ramach tego oprogramowania oferujemy zainteresowanym dostawę uniwersalnych pakietów programowych:

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

To doskonałe narzędzia pracy dla każdego. Aby z nich korzystać, nie trzeba być informatykiem! Zupełnie samodzielnie można tworzyć złożone systemy zarządzania przedsiębiorstwem, każdym przedsiębiorstwem; nawet najbardziej specyficzne uwarunkowania nie są przeszkodą.
To jednak jeszcze nie wszystko... Kiedy dotychczasowe problemy łatwo i szybko zostały rozwiązane — pojawiają się zupełnie nowe. Można wtedy bez kłopotów samemu udoskonalić dotychczasowy system!

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

składają się w zakładowe systemy płacowe, osobowe, finansowo-księgowe lub magazynowe. Korzystając z nich, z łatwością można prowadzić planowanie, kalkulacje i sprawozdawczość. Można też sporządzać kosztorysy i oferty, a nawet prowadzić „automatyczną" korespondencję czy redagować dowolne teksty. Można wreszcie skorzystać z już zgromadzonych zasobów na komputerze ODRA (pod nadzorem systemu GEORGE-3), wykorzystując komputer osobisty jako inteligentny terminal — stację lub emulator TTY.

Nowość:

Oferujemy system operacyjny kompatybilny z CP/M 2.2. dla mikrokomputerów ROBOTRON 5120/5130 oraz systemy finansowo-księgowe FK dla dowolnych mikrokomputerów

---

Szczegółowych informacji udziela:

CSK—Computer Studio Kajkowscy

81-505 GDYNIA ORŁOWO, ul. Balladyny 3B, tel. 29-00-18

EO/129/K/85